Witness Encryption from Garbled Circuit and Multikey Fully Homomorphic Encryption Techniques

Kamil Kluczniak[1,2]

[1] CISPA Helmholtz Center for Information Security
Saarland Informatics Campus
[2] Stanford University
kamil.kluczniak@{cispa.saarland,stanford.edu}

In a witness encryption scheme, to decrypt a ciphertext associated with an NP statement, the decrypter takes as input a witness testifying that the statement is in the language. When the statement is not in the language, then the message is hidden. Thus far, the only provably secure constructions assume the existence of indistinguishability obfuscation ($i\mathcal{O}$) and multilinear maps (MMaps).

We make progress towards building polynomially efficient witness encryption for NP without resorting to $i\mathcal{O}$ or MMaps. In particular, we give a witness encryption scheme from Yao's garbled circuit technique and a new type of fully homomorphic encryption (FHE) that we call annihilating. Interestingly, we require a version of the annihilating FHE that is circularly insecure, i.e., allows testing the presence of a key cycle. We prove our witness encryption's security from a novel assumption about our annihilating FHE. We formulate the assumption as an interplay between an annihilating FHE and ciphertexts modeled as ideal ciphers. We show a candidate (leveled) annihilating FHE built from a multikey variant of the BGV/BFV fully homomorphic cryptosystems.

1 Introduction

Witness Encryption (WE), introduced by Garg et al. [GGSW13], is a cryptosystem where the encryption algorithm Enc beside a message msg, takes as input a statement stmt from an NP language $\mathcal{L}$ associated with a polynomial-time checkable relation $\mathcal{R}$. To decrypt the message msg, the decryption algorithm Dec takes as input a witness wit such that $(\mathsf{stmt}, \mathsf{wit}) \in \mathcal{R}$. For security, we require that if $\mathsf{stmt} \notin \mathcal{L}$ (no witness exists), then it is infeasible to distinguish between potentially encrypted messages.

Besides offering very general access control and being an exciting primitive on its own, WE has numerous applications in cryptography. Among others, we can use WE to design identity and attribute-based encryption [GGSW13], secret sharing [KNY17], oblivious transfer [BGI+17], multiparty computation [GGHR14, GLS15], time-lock encryption [LJKW18], (optimally laconic) argument systems [KMN+14, BP15, FNV17, BISW18, BKP19], indistinguishability obfuscation for evasive functions [GKW17a, WZ17], or reusable garbled circuits and single key succinct functional encryption for Turing machines [GKP+13].

1.1 Contribution

Since its inception, provably secure witness encryption schemes for NP were constructed either from indistinguishability obfuscation [GGH+13] or from generic multilinear maps [GGSW13, GKP+13, GLW14, LJKW18].

Our main contribution is to introduce new techniques from which we can build witness encryption schemes for all NP with security reductions to novel assumptions. New design paradigms are desirable, as they might deepen the understanding of a primitive and lead to more efficient instantiations under different assumptions. The design paradigm behind our witness encryption construction follows a straightforward intuition on how such cryptosystem should look like:

> *A witness encryption is a special encoding of a boolean formula that we can decode gate-by-gate according to the witness until we decode the message from the output gate. Furthermore, it is hard to decode the encoding inconsistently with the formula, and in consequence, decrypt the message when no witness exists.*

Notably, our witness encryption scheme handles general formula satisfiability. Thus we can directly instantiate it for SAT related problems. Previous constructions, excluding $i\mathcal{O}$ based, were limited to the exact cover or subset sum problems. We reduce our schemes' security to the security of primitives not known to imply $i\mathcal{O}$ or MMaps. In particular, we introduce a new primitive called annihilating fully homomorphic encryption (AFHE). For our WE scheme, we assume the security of a particular interplay between AFHE and ideal ciphers. We call it ideal cipher annihilation security. The assumption shares some conceptual similarities to linear-only encryption [BISW17] or targeted malleability [BSW12], but comes with a novel spin. Interestingly we use a circularly insecure AFHE, in the sense that it is possible to detect key cycles. Previously such systems were constructed solely to show separations between semantic and circular security. Our work shows that we can view circularly insecure encryption as a useful tool to build advanced cryptographic primitives. We give a candidate instantiation of the AFHE scheme based on a multikey version of the BGV/BFV (leveled) fully homomorphic encryption schemes [Bra12, BV11, BGV12, FV12]. Then we compile it to a circularly insecure version via lockable obfuscation [GKW17a, WZ17, CVW18]. We conjecture that our candidate satisfies ideal cipher annihilating security. This conjecture constitutes the heuristic part of our contribution. However, we argue that potential attack vectors on the candidate with practical instantiations of the ideal cipher are very unlikely to work and could potentially lead to the development of techniques useful to build more efficient homomorphic schemes.

1.2 Overview of Our Techniques

The intuitive design paradigm mentioned in the previous section is the starting point to build our scheme.

Basic Scheme form Yao's Technique. Observe that to implement the idea of a boolean formula encoding, the first thing that comes to mind is to use Yao's garbled circuits [Yao86]. In particular, to build a basic witness encryption scheme, we can garble a formula (encrypt the gates), but the last garbled gate encrypts the message in place of the accepting label. Then, the witness encryption consists of the garbled circuit and all labels of input variables. We note that we do not use garbled circuits as a black-box primitive. We do not need to permute the garbled gates, and we do not rely on the standard properties of garbled circuits.

Let us illustrate the basic witness encryption scheme on the following boolean formula: $C(a,b) = (a\ \texttt{AND}_x\ 0)\ \texttt{AND}_z\ (b\ \texttt{AND}_y\ 0)$. Denote the output wire of $(a\ \texttt{AND}_x\ 0)$ as c and $(b\ \texttt{AND}_y\ 0)$ as d. Clearly, there is no valuation of $a, b \in \{0,1\}$ that $C(a,b) = 1$. Now, we build a witness encryption scheme for C as follows. The encrypter chooses keys K_a^0, K_b^0, K_a^1, K_b^1, K_c^0, K_d^0, K_c^1, $K_d^1 \in \{0,1\}^{\ell_{\underline{\mathsf{sk}}}}$ representing all value assignments of all wires in the circuit, where $\ell_{\underline{\mathsf{sk}}}$ is the key size. We call these keys labels. To encrypt a message msg, we create garbled gates following the truth tables of gates in the circuit. That is, the encrypter encodes the $\texttt{AND}_z$ gate: $\mathsf{ct}_z^{0,0} \leftarrow \mathsf{Enc}([K_c^0, K_d^0], \mathsf{reject})$, $\mathsf{ct}_z^{0,1} \leftarrow \mathsf{Enc}([K_c^0, K_d^1], \mathsf{reject})$, $\mathsf{ct}_z^{1,0} \leftarrow \mathsf{Enc}([K_c^1, K_d^0], \mathsf{reject})$, $\mathsf{ct}_z^{1,1} \leftarrow \mathsf{Enc}([K_c^1, K_d^1], \mathsf{msg})$. Analogously, the encrypter encodes the $\texttt{AND}_x$ and $\texttt{AND}_y$ gates: $\mathsf{ct}_x^0 \leftarrow \mathsf{Enc}([K_a^0], K_c^0)$, $\mathsf{ct}_x^1 \leftarrow \mathsf{Enc}([K_a^1], K_c^0)$, and $\mathsf{ct}_y^0 \leftarrow \mathsf{Enc}([K_b^0], K_d^0)$, $\mathsf{ct}_y^1 \leftarrow \mathsf{Enc}([K_b^1], K_d^0)$. The ciphertext includes all variable labels $K_a^0, K_b^0, K_a^1, K_b^1$ and all garbled gate ciphertexts $\mathsf{ct}_z^{0,0}$, $\mathsf{ct}_z^{0,1}$, $\mathsf{ct}_z^{1,0}$, $\mathsf{ct}_z^{1,1}$, ct_x^0, ct_x^1, ct_y^0, ct_y^1. Note that we do not include labels of internal wires. It is easy to see that, to decrypt the message msg, a decrypter would first need to obtain the labels corresponding to $c = 1$ and $d = 1$. That is, we would need to decrypt somehow K_c^1, K_d^1 from the garbling of the $\texttt{AND}_x$, and $\texttt{AND}_y$ gates. However, none of the $\texttt{AND}_x$ and $\texttt{AND}_y$ garbled gates contains any information about K_c^1 or K_d^1. Therefore, even though the decrypter has all variable labels, the keys K_c^1, K_d^1 are not decryptable from the garbled circuit.

Mixed Input Attack. Unfortunately, applying the garbled circuit technique works only for a narrow family of read-once formulas. That is, formulas where each variable is assigned to a single input wire. We can prove (see Appendix A) that a scheme as above yields an (extractable) witness encryption for read-once formulas in the ideal cipher model. However, for us, this is unsatisfactory as the NP-complete SAT problem requires formulas that read each variable polynomially many times. The problem is that we cannot force the adversary A, who is given all the variable labels, to evaluate the garbled circuit consistently. In other words, despite the formula being read-many, no mechanism prevents A from treating the garbled formula as read-once, which satisfiability is in P. It is easy to see the problem for example on the (non-satisfiable) formula $C(a) = (a\ \texttt{AND}_x\ 1)\ \texttt{AND}_z\ (a\ \texttt{NAND}_y\ 1)$. When garbling C and publishing all the labels for the variable a, an adversary A can decode all the internal wire keys corresponding to 1 by applying both 0 and 1 labels of a. In other words, from A's perspective, a garbling of C is equivalent to a garbling of the (satisfiable) read-once formula $C'(a,a') = (a\ \texttt{AND}_x\ 1)\ \texttt{AND}_z\ (a'\ \texttt{NAND}_y\ 1)$, with an additional variable a'.

Enforcing Consistency. Observe that whenever an adversary A learns all variable labels of the garbled circuit, A may always run the mixed input attack. Hence, our first step is to encrypt the variable labels with a fully homomorphic encryption scheme. This way, the adversary does not learn the labels but can still evaluate the garbled circuit homomorphically.

Now, to solve the mixed input issue, we build a special fully homomorphic encryption scheme that we call annihilating. In an annihilating fully homomorphic encryption (AFHE), we assign a set of tags to each ciphertext. When computing on such ciphertexts, the result is assigned the union of the tags. We can mark some tags as mutually annihilating and require that it is hard to perform a correct homomorphic computation on ciphertexts that contain mutually annihilating tags. Now we encrypt each variable labels from the garbling scheme with a unique tag, and mark tags that correspond to different valuations of the same variable as annihilating. This way, the AFHE "remembers" which variable labels a decrypter used to obtain an internal wire label. Finally, attempts to evaluate the decryption circuit of a garbled gate on ciphertexts with annihilating tags (corresponding to contradictory valuations of input variables) should result in a ciphertext holding random noise. Therefore, mixed input attacks should not be a useful strategy to decode the garbled circuit, and the only option left is to decode consistently.

Testing the Encrypted Output. We give more details on annihilating FHE in the next paragraph. The problem now is that an honest decrypter with a valid witness can obtain only an AFHE ciphertext of the message (the output of the last garbled gate) instead of the actual message. Moreover, as the FHE should be semantically secure, there seems to be no way to decode the message.

To resolve this issue, the first idea that comes to mind is to have the garbled circuit encode $\mathsf{AFHE.Dec}(\mathsf{sk}, \mathsf{msg})$ instead of the message. When homomorphically evaluating the garbled circuit, we get $\mathsf{AFHE.Enc}(\mathsf{sk}, \mathsf{AFHE.Dec}(\mathsf{sk}, \mathsf{msg})) = \mathsf{msg}$. Unfortunately, we do not know of any FHE scheme, for which $\mathsf{AFHE.Dec}(\mathsf{sk}, \mathsf{msg})$ would be a well-formed plaintext.

The next iteration of this idea is to have the garbled circuit encode sk and use an (annihilating) FHE that is semantically secure but breakable for key-dependent plaintexts. In other words, there is a distinguisher that detects key cycles [CL01]. Then upon homomorphically evaluating the garbled circuit, we can test whether the plaintext is sk (indicating that the message is 1) or something else (telling that it is 0).

We note that a key-dependent plaintext might result in the total break of the encryption scheme. However, the adversary must first homomorphically decrypt the accepting label to break the scheme. Until that event, all plaintexts are independent of the key. Furthermore, given that the formula is not-satisfiable, and the AFHE system prevents inconsistent computation, the adversary will never decrypt the accepting label.

Annihilating Fully Homomorphic Encryption. Let us first address the possibility of realizing AFHE using FHE with targeted malleability [BSW12].

Targeted malleability guarantees that an evaluator can compute only functions on ciphertexts specified in the FHE's public key. Therefore, at a high level, it may seem that targeted malleability immediately solves our problem, as we could limit the evaluator to functions consistent with the tags of the AFHE. Unfortunately, targeted malleability does not require the plaintext to be "destroyed" upon attempts to compute invalid functions. It merely requires the existence of a verification algorithm that judges whether a ciphertext came out of a consistent evaluation or not.

To build annihilating encryption, we put our focus on multikey FHE (MKFHE) schemes instead. MKFHE is a generalization of (single key) FHE introduced by López-Alt, Tromer, and Vaikuntanathan [LTV12], that allows computation on ciphertexts created using different secret keys. Many current MKFHE candidates [CM15, MW16, PS16, BP16, CCS19, CDKS19] require to publish special evaluation keys to allow computation on encrypted data of distinct origin. Thus our idea is as follows: *We associate a secret key with a tag. If we want to mark two tags as annihilating, we simply do not publish the special evaluation keys corresponding to those tags.*

To simplify exposition, we describe our idea for the basic scheme over a ring $\mathcal{R}_q$ and the two-key case. Also, we omit to denote the way messages and errors are encoded. To encrypt $\mathfrak{m}_1, \mathfrak{m}_2 \in \mathcal{R}_q$ under keys $\mathfrak{s}_1, \mathfrak{s}_2 \in \mathcal{R}_q$, we use the Regev cryptosystem [Reg05]. Specifically, we choose $\mathfrak{a}_i \xleftarrow{\$} \mathcal{R}_q$, $\mathfrak{e}_i \xleftarrow{\$} \mathcal{X}$, and set $\mathbf{c}_i = [\mathfrak{b}_i, \mathfrak{a}_i] = [-\mathfrak{a}_i \cdot \mathfrak{s}_i + \mathfrak{m}_i + \mathfrak{e}_i, \mathfrak{a}_i]$ for $i \in \{1, 2\}$. To decrypt each ciphertext with respect to a secret key $\mathfrak{s}_i$ we run $\mathfrak{m}_i \leftarrow \lfloor \langle \mathbf{c}_i, [1, \mathfrak{s}_i] \rangle \rceil$, where $\lfloor . \rceil$ is the appropriate rounding function. Note that we can generalize the decryption function to handle multiple secret keys as $\langle \hat{\mathbf{c}}_i, [1, \mathfrak{s}_1, \mathfrak{s}_2] \rangle$ and extend $\mathbf{c}_i$ to $\hat{\mathbf{c}}_i$, where we set $\hat{\mathbf{c}}_1 \leftarrow [\mathfrak{b}_1, \mathfrak{a}_1, 0]$, and $\hat{\mathbf{c}}_2 \leftarrow [\mathfrak{b}_2, 0, \mathfrak{a}_2]$. This way we can "lift" ciphertexts without increasing the noise and add the ciphertexts as $\hat{\mathbf{c}}_1 + \hat{\mathbf{c}}_2$.

Multiplication is a bit more involved. In BGV/BFV type cryptosystems [Bra12, BV11, BGV12, FV12], the first step towards multiplying two ciphertexts is to compute the Kronecker product of the ciphertexts $\hat{\mathbf{c}}_1$ and $\hat{\mathbf{c}}_2$ under secret key $\hat{\mathbf{s}} = [1, \mathfrak{s}_1, \mathfrak{s}_2]$. Note that $\langle \hat{\mathbf{c}}_1, \hat{\mathbf{s}} \rangle \cdot \langle \hat{\mathbf{c}}_2, \hat{\mathbf{s}} \rangle = \langle \hat{\mathbf{c}}_1 \otimes \hat{\mathbf{c}}_2, \hat{\mathbf{s}} \otimes \hat{\mathbf{s}} \rangle$, what follows from the mixed-product property of the Kronecker product. So $\tilde{\mathbf{c}} = \hat{\mathbf{c}}_1 \otimes \hat{\mathbf{c}}_2$ is a ciphertext of the product of the two messages, under the key $\tilde{\mathbf{s}} = \hat{\mathbf{s}} \otimes \hat{\mathbf{s}} = [1, \mathfrak{s}_1, \mathfrak{s}_2, \mathfrak{s}_1, \mathfrak{s}_1^2, \mathfrak{s}_1 \cdot \mathfrak{s}_2, \mathfrak{s}_2, \mathfrak{s}_1 \cdot \mathfrak{s}_2, \mathfrak{s}_2^2]$. Unfortunately, we cannot keep multiplying using the Kronecker product as the ciphertexts' size grows exponentially. Hence we need to map the ciphertext back to the form that decrypts with $\hat{\mathbf{s}}$. This process is called relinearization (or keyswitchnig in general), and requires to publish special encryptions of $\mathfrak{s}_i, \mathfrak{s}_i^2$ for $i \in \{1, 2\}$, and $\mathfrak{s}_1 \cdot \mathfrak{s}_2$. Roughly speaking, using the relinearization keys, one homomorphically decrypts $\tilde{\mathbf{c}}$. The crucial observation that we want to point to is that without having the encryptions of $\mathfrak{s}_1 \cdot \mathfrak{s}_2$, we end up with a standard BGV/BFV cryptosystem that is not multikey. It seems to be infeasible to relinearize the extended ciphertext completely and remove the part dependent on $\mathfrak{s}_1 \cdot \mathfrak{s}_2$. Hardness to perform multikey multiplication is, in some way, analogical to the hardness of performing non-linear operations in the linear-only encryption of Boneh et al. [BISW17]. In particular, the linear-only cryptosystem

[BISW17], is basically a BGV/BFV scheme without any relinearization keys (just as symmetric Regev encryption), with only one additional tweak to make the ciphertext space sparse needed to justify extractability that in our setting is not required.

To sum up, we construct our candidate annihilating FHE from a version of BGV/BFV that is symmetric and multikey. Each secret key is associated with a unique tag, and for annihilating tags/keys, we do not publish the part of the relinearization key holding $\mathfrak{s}_1 \cdot \mathfrak{s}_2$. Finally, we conjecture that without encryptions of $\mathfrak{s}_1 \cdot \mathfrak{s}_2$ it is infeasible to compute non-linear functions on corresponding multikey ciphertexts. The assumption needed for our WE is stated in conjunction with ideal ciphers that, in practice, can be instantiated by block-ciphers (e.g., AES256). Roughly speaking, the assumption says that the query patterns to the ideal cipher given real AFHE ciphertexts should be as in an ideal system. In the ideal system, we don't allow queries to an ideal cipher on data encrypted under annihilating tags, but we allow queries to the ideal cipher on data encrypted under non-annihilating tags. The output of a successful query is an AFHE encryption of what the oracle returns. We note, however, that as linear-only encryption [BISW17] or extractable FHE [BC12], our assumption is not falsifiable.

Circularly Insecure Annihilating Fully Homomorphic Encryption. Fortunately, there are candidate constructions for semantically secure encryption schemes that are not circular secure [AP16, KW16, GKW17b]. We put our attention on the works from Goyal, Koppula, and Waters [GKW17a], and Wichs and Zirdelis [WZ17], who introduced a primitive called lockable obfuscation[3] and used it to build cycle testers that distinguish whether a chain of ciphertexts contains a key cycle or not. Lockable obfuscation is a mechanism that takes as input a circuit C and a lock value lock and outputs an obfuscated circuit $\tilde{C}$. An evaluator can run the obfuscated circuit on any input x, and $\tilde{C}$ returns 1 if $C(x) = \mathsf{lock}$, and $\perp$ otherwise. For security, $\tilde{C}$ reveals no information on C (virtual black-box security) assuming that lock has sufficient entropy even given the circuit C. Importantly we can build lockable obfuscation assuming hardness of the (subexponential) learning with errors problem [GKW17a, WZ17, CVW18]. Furthermore, lockable obfuscation is not known to imply $i\mathcal{O}$ or MMaps. To build our cycle tester, we use the generic compiler from [GKW17a]. In a nutshell, the compiler extends the secret key of an encryption scheme with a uniformly random lock and publishes a lockable obfuscation where $C(x) = \mathsf{Dec}(\mathsf{sk}, x)$.

Note that our approach shares some similarities to the concept of zero-testing required by multilinear maps, in that there exists a plaintext for which we can perform a test. However, in contrast to zero-testing, the testable message has high entropy in our case, and we know secure methods to implement such testers.

An important caveat is that we cannot show that annihilating security is preserved when given a lockable obfuscation. While the lockable obfuscation

[3] Wichs and Zirdelis [GKW17a] call the primitive distributional virtual black-box obfuscation for compute and compare programs. We refer to the primitive shortly as lockable obfuscation as in [WZ17].

may not reveal any information on the circuit, it does not guarantee that we cannot use it to perform homomorphic computation. For our candidate AFHE construction, we chose to instantiate the lockable obfuscation with the concrete construction from [CVW18]. Since the construction is based on noisy encodings [GGH15, CVW18] and is designed for circuits in NC^1 only, it seems unlikely that obfuscated programs might help in any way to break the annihilating security of our candidate.

1.3 Previous Work

Garg, Gentry, Sahai, and Waters [GGSW13] introduced the concept of witness encryption and showed a candidate construction for the exact cover problem from generic multilinear maps. Several other works [GKP+13, LJKW18] built upon [GGSW13], and proposed (extractable) witness encryption schemes, or witness pseudorandom functions [Zha16], based on similar assumptions. To be more specific, these assumptions rely on generic multilinear maps and depend on the exact cover or subset sum problems. Gentry, Lewko, and Waters [GLW14] showed the first construction of a witness encryption scheme for NP from instance independent assumptions, albeit their candidate still requires multilinear maps.

Recently Barta et al. [BIOW20] showed a witness encryption scheme for a variant of the problem of approximating the minimum distance of a linear code (GapMDP) assuming only generic groups. However, it remains an open question whether GapMDP for certain specific parameters is NP-hard. To be more specific, they only hypothesize that there exists a deterministic polynomial-time reduction from NP to their version of GapMDP. Unfortunately, we do not know whether such reduction exists and whether the witness encryption works for all problems in NP or not.

Bartusek et al. [BIJ+20] design witness encryption from new algebraic structures, and Chen et al. [CVW18] show a construction from lattice-based techniques. Both candidates are conjectured to be secure. However, there is also no attempt made to provide a security reduction for those schemes.

Garg et al. [GGH+13] observed that $i\mathcal{O}$ for NC^1 is enough to build witness encryption for NP. $i\mathcal{O}$ with reductionist security analysis, until very recently was constructed mainly from multilinear maps, or assuming the existence of PRG's with special properties [LT17, AJL+19, Agr19, JLMS19, BHJ+19]. Recently, Jain, Lin, and Sahai showed a candidate assuming well studied assumptions and the existence of PRGs in NC^0 with polynomial stretch. Brakerski et al. [BDGM20a] followed by a sequence of works [GP20, BDGM20b, WW20], show candidates for non-trivial exponentially efficient $i\mathcal{O}$, called X$i\mathcal{O}$, assuming certain circular security assumptions. Plugging the X$i\mathcal{O}$ into the framework of Lin et al. [LPST16], we obtain polynomially efficient $i\mathcal{O}$, assuming additionally (subexponential) security of the LWE problem. Brakerski et al. [BJK+18] proposed a nontrivial exponentially efficient witness encryption. However, no framework analogous to [LPST16] is known for witness encryption.

Importantly building WE without $i\mathcal{O}$ or MMaps but with a security reduction to instance independent assumptions is a long-standing open problem.

2 Preliminaries

Notation. We denote as $\mathbf{A}[i,j]$ the entry of matrix $\mathbf{A}$ in the ith row and jth column. When addressing the ith row of a matrix $\mathbf{A}$ we write $\mathbf{A}[i,*]$. Analogically, we address the jth column of a matrix $\mathbf{A}$ as $\mathbf{A}[*,j]$. We denote $\mathbf{v}^\top \in \mathbb{R}^{1\times n}$, the transposition of $\mathbf{v} \in \mathbb{R}^{n\times 1}$. For brevity, we denote as $[n]$ the vector $[i]_{i=1}^{n}$. By default we abbreviate row vectors $\mathcal{R}^{1\times n}$ as $\mathcal{R}^n$. For a function f we write $f([x_i]_{i=1}^n)$ instead of $f(x_1,\ldots,x_n)$. We denote $||\mathbf{a}||_p = (\sum_{i=1}^{p} |a_i|^p)^{1/p}$ the p-norm of a vector $\mathbf{a} \in \mathbb{R}^n$, where $|x|$ denotes the absolute value of $x \in \mathbb{R}$. For polynomials of degree N in $\mathcal{R}$, we compute the p-norm by taking its coefficient vector. We denote any positive polynomial as $\mathsf{poly}(.)$. Finally, we denote as $\mathsf{negl}(.)$ any negligible function. That is, for any positive polynomial $\mathsf{poly}(.)$ there exists $c \in \mathbb{N}$ such that for all $\lambda \geq c$ we have $|\mathsf{negl}(\lambda)| \leq \frac{1}{\mathsf{poly}(\lambda)}$.

Circuits and Formulas. We give the notation and terminology for circuits.

Definition 1 (Circuit). *A circuit $C : \Lambda^\eta \mapsto \Lambda^\nu$, over the alphabet Λ is a directed acyclic graph, where vertices are called gates and edges are called wires. Gates are associated with a function $f \in \mathcal{F}$ from the family of functions $\mathcal{F} : \Lambda^\tau \mapsto \Lambda^\delta$, where τ is the number of incoming wires called fan-in, and δ is the number of outgoing wires called the fan-out of the gate. If a gate does not have any incoming wires outgoing for another gate, such a gate is called an input gate, and its incoming wires are called input wires. Each input wire is associated with a variable x_i indexed by $i \in [\eta]$. If a gate has outgoing wires which are not incoming for any other gate, such gates are called output gates, and the outgoing wires are called output wires. Any other gate (resp. wire) is called an internal gate (resp. wire). We call the number of gates in a circuit C the size of the circuit and denote it as $|C|$. Finally we access the jth symbol in the alphabet as $\Lambda[j]$.*

Definition 2 (Formula). *A formula is a circuit $C : \Lambda^\eta \mapsto \Lambda$ over $\mathcal{F}$ arrangeable to the form of a rooted tree. Equivalently each gate has fan-out at most one, and each internal wire is input to only a single gate. We say that a formula is read-once if every variable is assigned to only a single input wire.*

Definition 3 (Satisfiability). *Let $C : \Lambda^\eta \mapsto \Lambda$ be a circuit over an alphabet Λ with η input variables and a single output wire. We define the circuit satisfiability problem, given (C, v) decide whether there exists $x \in \Lambda^\eta$ such that $C(x) = v$, where $v \in \Lambda$ is called the accepting symbol.*

Due to the Cook-Levin Theorem, it is widely known that the satisfiability of boolean formulas (SAT) is NP-complete.

Notation for Trees. Finally, in this paper, we represent formulas as trees. Here we list the notation we use to operate on trees.

We treat a vertex g as an object with internal variables and methods. For a vertex g, we define $\mathsf{g.child}(j)$ the method which on input an index $j \in [\tau]$ outputs

the jth child of g. If g is an input gate, $\mathsf{g.child}(j)$ outputs null. If a gate is an input gate, then $\mathsf{g.wireVar}(j)$ outputs the index $i \in [\eta]$ of the variable assigned to the jth incoming wire of the gate. By $\mathsf{g.fan\text{-}in}()$, we denote the function that outputs the number of incoming wires to that gate.

Encryption Schemes and Ideal Cipher Model. We give our syntax for symmetric encryption and recall the ideal cipher model [Sha49]. Recall that the ideal cipher model and the random oracle model are equivalent [CPS08]. Moreover, the ideal cipher or the random permutation models are often used to argue the security of dual-key ciphers for garbled circuit schemes [BHR12].

Definition 4 (Symmetric Key Encryption). *Let λ be a security parameter and let $\ell_{\underline{\mathsf{sk}}}, \ell_{\underline{\mathsf{ct}}}, \tau = \mathsf{poly}(\lambda)$. An encryption scheme $\mathsf{SKE} = (\mathsf{Enc}, \mathsf{Dec})$ consists of an encryption algorithm Enc and decryption algorithm Dec with the following syntax.*

$\mathsf{Enc}([\underline{K}_k]_{k=1}^{\tau}, \mathsf{msg})$**:** *Takes as input secret keys $[\underline{K}_k]_{k=1}^{\tau}$ and a message $\mathsf{msg} \in \{0,1\}^{\ell_{\underline{\mathsf{sk}}}}$, and outputs a ciphertext $\underline{\mathsf{ct}} \in \{0,1\}^{\ell_{\underline{\mathsf{ct}}}}$.*

$\mathsf{Dec}([\underline{K}_k]_{k=1}^{\tau}, \underline{\mathsf{ct}})$**:** *This deterministic algorithm takes as input keys $[\underline{K}_k]_{k=1}^{\tau} \in \{0,1\}^{\tau \cdot \ell_{\underline{\mathsf{sk}}}}$ and a ciphertext $\underline{\mathsf{ct}}$. The algorithm outputs msg.*

(Perfect) Correctness: *We say that $\mathsf{SKE} = (\mathsf{Enc}, \mathsf{Dec})$ is correct, if for all security parameters λ and all $\ell_{\underline{\mathsf{sk}}}, \ell_{\underline{\mathsf{ct}}}, \tau = \mathsf{poly}(\lambda)$, all keys $[\underline{K}_k]_{k=1}^{\tau} \in \{0,1\}^{\tau \cdot \ell_{\underline{\mathsf{sk}}}}$ and all messages $\mathsf{msg} \in \{0,1\}^{\ell_{\underline{\mathsf{sk}}}}$, the following holds.*

$$\mathsf{Dec}([\underline{K}_k]_{k=1}^{\tau}, \mathsf{Enc}([\underline{K}_k]_{k=1}^{\tau}, \mathsf{msg})) = \mathsf{msg}.$$

Ideal Cipher Model: *We model $\mathsf{SKE} = (\mathsf{Enc}, \mathsf{Dec})$ for all security parameters λ, and all $\ell_{\underline{\mathsf{sk}}}, \ell_{\underline{\mathsf{ct}}}, \tau = \mathsf{poly}(\lambda)$ as follows. On input a vector of binary strings $[\underline{K}_k]_{k=1}^{\tau} \in \{0,1\}^{\tau \cdot \ell_{\underline{\mathsf{sk}}}}$ and $\mathsf{msg} \in \{0,1\}^{\ell_{\underline{\mathsf{sk}}}}$ to Enc the oracle chooses $\underline{\mathsf{ct}} \xleftarrow{\$} \{0,1\}^{\ell_{\underline{\mathsf{ct}}}}$, stores $([\underline{K}_k]_{k=1}^{\tau}, \underline{\mathsf{ct}}, \mathsf{msg})$ for future calls and returns $\underline{\mathsf{ct}}$. On input a set of strings $([\underline{K}_k]_{k=1}^{\tau}, \underline{\mathsf{ct}}) \in \{0,1\}^{\tau \cdot \ell_{\underline{\mathsf{sk}}}}$ to Dec the oracle looks up for the tuple $([\underline{K}_k]_{k=1}^{\tau}, \underline{\mathsf{ct}}, \mathsf{msg})$ in its internal memory and returns msg if such tuple is present, otherwise it chooses msg uniformly at random and stores the query for a future call.*

Remark 1. Representing the secret key as a vector serves only a notational purpose, and can trivially be realized by setting a single key $K = \underline{K}_1 || \ldots || \underline{K}_\tau$ where $||$ denotes concatenation. In Section 3.2, we discuss potential instantiations from block ciphers or key derivation functions.

Witness Encryption. We recall the definition of witness encryption introduced by Garg, Gentry, Sahai, and Waters [GGSW13].

Definition 5 (Witness Encryption). *A witness encryption scheme WE consists of algorithms $(\mathsf{Enc}, \mathsf{Dec})$ with the following syntax.*

$\mathsf{Enc}(\lambda, \mathsf{stmt}, \mathsf{msg})$: *This* **PPT** *algorithm takes as input a security parameter* λ*, a statement* stmt *and message* msg *and outputs a ciphertext* ct*.*

$\mathsf{Dec}(\mathsf{stmt}, \mathsf{wit}, \mathsf{ct})$: *This deterministic algorithm takes as input a statement* stmt*, a witness* wit *and a ciphertext* ct*, and outputs a message* msg *or* $\perp$*.*

Correctness: *We say that a witness encryption scheme* $\mathsf{WE} = (\mathsf{Enc}, \mathsf{Dec})$ *is correct if for all languages* $\mathcal{L} \in \mathsf{NP}$ *with witness relation* $\mathcal{R}$*, all security parameters* $\lambda \in \mathbb{N}$*, all statements* $\mathsf{stmt} \in \mathcal{L}$*, all witnesses* wit*, s.t.,* $\mathcal{R}(\mathsf{stmt}, \mathsf{wit}) = 1$*, and messages* $\mathsf{msg} \in \mathcal{M}$ *where* $\mathcal{M}$ *is the message space, we have*

$$\Pr[\mathsf{Dec}(\mathsf{stmt}, \mathsf{wit}, \mathsf{Enc}(\lambda, \mathsf{stmt}, \mathsf{msg})) \neq \mathsf{msg}] \leq \mathsf{negl}(\lambda),$$

where the probability is taken over the random coins of Enc*. We say that the scheme is perfectly correct if the above probability is always zero.*

Security: *A witness encryption scheme* $\mathsf{WE} = (\mathsf{Enc}, \mathsf{Dec})$ *is secure if for all security parameters* $\lambda \in \mathbb{N}$*, all languages* $\mathcal{L} \in \mathsf{NP}$*, all* $\mathsf{stmt} \notin \mathcal{L}$ *and all* **PPT** *adversaries* $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2)$*,*

$$\left| \Pr\left[\mathsf{A}_2(\mathsf{ct}, \mathsf{st}) = b \ : \begin{array}{c} (\mathsf{msg}_0, \mathsf{msg}_1, \mathsf{st}) \leftarrow \mathsf{A}_1(\mathsf{stmt}, \lambda); \\ b \xleftarrow{\$} \{0,1\}, \mathsf{ct} \leftarrow \mathsf{Enc}(\lambda, \mathsf{stmt}, \mathsf{msg}_b) \end{array} \right] - \frac{1}{2} \right| \leq \mathsf{negl}(\lambda),$$

where the probability is over the random coins of Enc *and random choice of* $b \in \{0,1\}$*.*

Lockable Obfuscation. As a building block, we use lockable obfuscation introduced by Goyal, Koppula, and Waters [GKW17a], and independently by Wichs and Zirdelis [WZ17].

Definition 6 (Lockable Obfuscation). *Let* $C = \{\mathcal{C}_{\eta,\nu,\xi}\}_{\eta,\nu,\xi \in \mathbb{N}}$ *be a family of circuits with* η *bit input,* ν *bit output and size* $\xi = |C|$*. A lockable obfuscation scheme* $\mathsf{lockObf} = (\mathsf{Obf}, \mathsf{Eval})$ *consists of an obfuscation algorithm* Obf *and an evaluation algorithm* Eval *with the following syntax.*

$\mathsf{Obf}(\lambda, C, \mathsf{lock})$**:** *This* **PPT** *algorithm takes as input a security parameter* $\lambda \in \mathbb{N}$*, a circuit* $C \in \mathcal{C}_{\eta,\nu,\xi}$*, where* $\eta, \nu, \xi = \mathsf{poly}(\lambda)$*, and a lock string* $\mathsf{lock} \in \{0,1\}^\nu$*. The algorithm outputs an obfuscated circuit* $\tilde{C}$*.*

$\mathsf{Eval}(\tilde{C}, \mathbf{x})$: *This deterministic algorithm takes as input an obfuscated circuit* $\tilde{C}$ *and input* $\mathbf{x} \in \{0,1\}^\eta$*, and outputs* 1 *or* $\perp$*.*

Functionality: *For all* $\lambda \in \mathbb{N}$*, all* $\eta, \nu, \xi, \ell_{\underline{\mathsf{sk}}} = \mathsf{poly}(\lambda)$*, all* $C \in \mathcal{C}_{\eta,\nu,\xi}$*, if* $\tilde{C} \leftarrow \mathsf{Obf}(\lambda, C, \mathsf{lock})$*, where* $\mathsf{lock} \in \{0,1\}^\nu$*, then*

$$\Pr[\mathsf{Eval}(\tilde{C}, \mathbf{x}) = 1] \geq 1 - \mathsf{negl}(\lambda),$$

given that $\mathbf{x} \in \{0,1\}^\eta$ *is such that* $C(\mathbf{x}) = \mathsf{lock}$*. The probability is over random coins of* Obf*. We say that the scheme is perfectly correct if the above probability is always one.*

Distributional Virtual Black-Box (DVBB) Security: *For all $\lambda \in \mathbb{N}$, all η, ν, ξ, $\ell_{\underline{\mathsf{sk}}} = \mathsf{poly}(\lambda)$, and for all* **PPT** *adversaries $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2)$, there exists a* **PPT** *simulator Sim, such that:*

$$\left| \Pr \left[\mathsf{A}_2(\tilde{C}_b, \mathsf{st}) = b : \begin{array}{c} (C, \mathsf{st}) \leftarrow \mathsf{A}_1(\lambda), C \in \mathcal{C}_{\eta,\nu,\xi}, \\ b \xleftarrow{\$} \{0,1\}, \mathsf{lock} \xleftarrow{\$} \{0,1\}^\nu, \\ \tilde{C}_0 \leftarrow \mathsf{Obf}(\lambda, C, \mathsf{lock}), \\ \tilde{C}_1 \leftarrow \mathsf{Sim}(\lambda, \eta, \nu, \xi) \end{array} \right] - \frac{1}{2} \right| \leq \mathsf{negl}(\lambda),$$

where the probability is over the choice of b and lock, and the random coins of Obf and Sim.

Learning With Errors. We recall the learning with errors assumption by Regev [Reg05]. We use generalized LWE as in [BGV12], to capture LWE and ring LWE at once. We extend the notation even further to manage multiple keys within a single GLWE tuple.

Definition 7 (Generalized Learning With Errors). *For a security parameter λ, let $n = \mathsf{poly}(\lambda)$ be an integer dimension, $N = \mathsf{poly}(\lambda)$ be a power of 2, $q = \mathsf{poly}(\lambda) \geq 2$. Furthermore let $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$ and $\mathcal{R}_q = \mathcal{R}/q\mathcal{R}$, and let $\mathcal{X}$ be a distribution over $\mathcal{R}$. For all $i \in [d]$ let $\mathbf{a}_i \xleftarrow{\$} \mathcal{R}_q^n$, $\mathbf{s}_i \xleftarrow{\$} \mathcal{R}_q^n$ and $\mathfrak{e}_i \xleftarrow{\$} \mathcal{X}$. We define a Generalized Learning With Errors (GLWE) sample of a message $\mathfrak{m} \in \mathcal{R}_q$ with respect to $[\mathbf{s}_i]_{i=1}^d$, as $\mathsf{GLWE}_{\mathcal{X},n,N,q}([\mathbf{s}_i]_{i=1}^d, \mathfrak{m}) = [-\sum_{i=1}^d \langle \mathbf{a}_i, \mathbf{s}_i \rangle + \mathfrak{e}_i + \mathfrak{m}, [\mathbf{a}_i]_{i=1}^d]$. The $\mathsf{GLWE}_{\mathcal{X},n,N,q}$ problem is to distinguish the following two distributions: In the first distribution, one samples $[\mathfrak{b}^{(j)}, [\mathbf{a}_i^{(j)}]_{i=1}^d]$ uniformly from $\mathcal{R}_q^{n \cdot d+1}$ for $j \in [\mathsf{poly}(\lambda)]$. In the second distribution one first draws $[\mathbf{s}_i]_{i=1}^d \xleftarrow{\$} \mathcal{R}_q^{n \cdot d}$ and then samples $\mathsf{GLWE}_{\mathcal{X},n,N,q}([\mathbf{s}_i]_{i=1}^d, 0) = [\mathfrak{b}^{(j)}, [\mathbf{a}_i^{(j)}]_{i=1}^d] \in \mathcal{R}_q^{n \cdot d+1}$. The $\mathsf{GLWE}_{\mathcal{X},n,N,q}$ assumption is that the probability of distinguishing the distributions is at most $\mathsf{negl}(\lambda)$.*

Definition 8 (LWE and RLWE samples). *We denote a Learning With Errors (LWE) sample as $\mathsf{LWE}_{\mathcal{X},n,q}(\mathbf{s}, m) = \mathsf{GLWE}_{\mathcal{X},n,1,q}(\mathbf{s}, m)$, which is a special case of a GLWE sample with $N = 1$. Similarly we denote a Ring-Learning with Errors (RLWE) sample as $\mathsf{RLWE}_{\mathcal{X},N,q}(\mathfrak{s}, \mathfrak{m}) = \mathsf{GLWE}_{\mathcal{X},1,N,q}(\mathfrak{s}, \mathfrak{m})$ which is the special case of an GLWE sample with $n = 1$.*

Definition 9 (Decomposition Gadget). *We call $\mathbf{g}_{\ell,B} = [B^{i-1}]_{i=1}^\ell \in \mathbb{N}^\ell$ for some $\ell, B \in \mathbb{N}$, the powers-of-B vector. We define the decomposition function $\mathsf{G}_{B,q}^{-1} : \mathcal{R}_q \mapsto \mathcal{R}_q^\ell$, where $\ell = \lceil \log_B q \rceil$, that on input a ring element $\mathfrak{a} \in \mathcal{R}_q$ outputs a vector $\mathbf{y} = [\mathfrak{a}_1, \ldots, \mathfrak{a}_\ell]$ such that $\mathfrak{a} = \sum_{i=1}^\ell \mathfrak{a}_i \cdot \mathbf{g}[i]$.*

3 Annihilating Fully Homomorphic Encryption

In this section, we introduce annihilating fully homomorphic encryption and show a plausible candidate in the leveled setting. For clarity, we denote AFHE keys/ciphertext with overline and symmetric keys/ciphertexts with an underline.

Definition 10 (Annihilating FHE). *An annihilating fully homomorphic encryption* AFHE *consists of algorithms* $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *with the following syntax.*

$\mathsf{Setup}(\lambda, \kappa, \mathcal{K})$**:** *This* **PPT** *algorithm takes as input a security parameter* λ*, a number of keys* $\kappa \in \mathsf{poly}(\lambda)$ *and a set of tags indicating which keys are annihilating* $\mathcal{K} \subseteq [\kappa] \times [\kappa]$ *such that if* $(i,j) \in \mathcal{K}$*, then* $(j,i) \in \mathcal{K}$*. The algorithm outputs a secret key* $\overline{\mathsf{sk}}$*, and an evaluation key* $\overline{\mathsf{ek}}$*.*

$\mathsf{Enc}(\overline{\mathsf{sk}}, \mathcal{T}, \mathsf{msg})$**:** *This* **PPT** *algorithm takes as input a secret key* $\overline{\mathsf{sk}}$*, a set of tags* $\mathcal{T} \subseteq [\kappa]$ *and a message* msg*, and returns a ciphertext* $\overline{\mathsf{ct}}$*.*

$\mathsf{Eval}([\overline{\mathsf{ct}}_i]_{i=1}^m, \overline{\mathsf{ek}}, C)$**:** *Given as input a set of ciphertexts* $[\overline{\mathsf{ct}}_i]_{i=1}^m$*, an evaluation key* $\overline{\mathsf{ek}}$ *and a circuit* C*, the algorithm outputs a ciphertext* $\overline{\mathsf{ct}}$*.*

$\mathsf{Dec}(\overline{\mathsf{sk}}, \overline{\mathsf{ct}})$**:** *This deterministic algorithm given a secret key* $\overline{\mathsf{sk}}$ *and a ciphertext* $\overline{\mathsf{ct}}$*, outputs a message* msg*.*

Correctness: *Let* $\lambda \in \mathbb{N}$ *and* $\kappa = \mathsf{poly}(\lambda)$*. Let* $\mathcal{K} \subseteq [\kappa] \times [\kappa]$ *be a set of annihilating tags, such that if* $(i,j) \in \mathcal{K}$*, then* $(j,i) \in \mathcal{K}$*, and* $\overline{\mathcal{K}} = ([\kappa] \times [\kappa]) \setminus \mathcal{K}$ *be the set of non-annihilating tags. We say that* $\mathsf{AFHE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *over some message space* $\mathcal{M}$ *is correct if for all circuits* C *with* η *input variables over* $\mathcal{M}$*, all* $[m_i]_{i=1}^\kappa \in \mathcal{M}^\kappa$*, all* $\mathcal{K}$ *and* $\overline{\mathcal{K}}$ *as defined above, all* $\mathcal{T} \subseteq [\kappa]$ *such that* $\eta = |\mathcal{T}|$ *and for all* $(i,j) \in \mathcal{T}$ *we have* $(i,j) \in \overline{\mathcal{K}}$*, we have*

$$\Pr\left[\mathsf{Dec}(\overline{\mathsf{sk}}, \overline{\mathsf{ct}}) = C([m_i]_{i\in\mathcal{T}}) : \begin{array}{c}(\overline{\mathsf{sk}}, \overline{\mathsf{ek}}) \leftarrow \mathsf{Setup}(\lambda, \kappa, \mathcal{K}),\\ [\overline{\mathsf{ct}}_i \leftarrow \mathsf{Enc}(\overline{\mathsf{sk}}, \{i\}, m_i)]_{i\in\mathcal{T}},\\ \overline{\mathsf{ct}} = \mathsf{Eval}([\overline{\mathsf{ct}}_i]_{i\in\mathcal{T}}, \overline{\mathsf{ek}}, C)\end{array}\right] \geq 1 - \mathsf{negl}(\lambda)$$

where the probability is over the random coins of Setup *and* Enc*.*

Definition 11 (Cycle Testing). *We define an additional algorithm,* $\mathsf{CycleTest}$*, to have the following syntax.*

$\mathsf{CycleTest}([\mathsf{ek}_k]_{k=1}^n, [\mathsf{ct}_k]_{k=1}^n)$**:** *On input a vector of evaluation keys* $[\mathsf{ek}_k]_{k=1}^n$ *and a vector of ciphertexts* $[\mathsf{ct}_k]_{k=1}^n$*, outputs a bit* $b \in \{0,1\}$*.*

We say that $\mathsf{AFHE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec}, \mathsf{CycleTest})$ *is an annihilating fully homomorphic encryption scheme with an* n*-cycle tester for a function* $\mathsf{F} : \mathcal{S} \mapsto \mathcal{M}$*, where* $\mathcal{M}$ *denotes the message space and* $\mathcal{S}$ *the secret key space, if for all* $\lambda \in \mathbb{N}$*,* $\kappa = \mathsf{poly}(\lambda)$*,* $\mathcal{K} \subseteq [\kappa] \times [\kappa]$ *such that if* $(i,j) \in \mathcal{K}$*, then* $(j,i) \in \mathcal{K}$*, all* $\mathcal{T}_k \subseteq [\kappa]$*, all* $[(\overline{\mathsf{sk}}_k, \overline{\mathsf{ek}}_k) \leftarrow \mathsf{Setup}(\lambda, \kappa, \mathcal{K})]_{k=1}^n$*, and* $[\overline{\mathsf{ct}}_k \leftarrow \mathsf{Enc}(\overline{\mathsf{sk}}_k, \mathcal{T}_k, \mathsf{F}(\overline{\mathsf{sk}}_{(k \mod n)+1}))]_{k=1}^n$*, we have*

$$\Pr[\mathsf{CycleTest}([\overline{\mathsf{ek}}_k]_{k=1}^n, [\overline{\mathsf{ct}}_k]_{k=1}^n) = 1] \geq 1 - \mathsf{negl}(\lambda),$$

where he probability is over random coins of Setup *and random coins of* Enc*.*

3.1 How to Define Annihilation Security

We define semantic security in the usual way, but we omit it here as we do not use semantic security explicitly in our application. However, for completeness, we devise semantic security to Appendix C.

Modeling that computing on annihilating tags "destroys" the plaintext turns out to be complicated. We decided to model how annihilating encryption should interplay with ideal ciphers. We define two experiments. In the real experiment RealExp, an adversary A is given pre-computed ciphertexts of an ideal cipher SKE, and AFHE encryptions of SKE secret keys. In the ideal experiment IdealExp, the adversary obtains uniformly random values instead of real AFHE encryptions, and access to an oracle Eval. The Eval oracle representing the homomorphic evaluation of SKE.Dec gets as input a vector of AFHE ciphertexts and a SKE ciphertext. If the AFHE ciphertexts are non-annihilating, then Eval queries SKE.Dec on the plaintexts they encrypt. Otherwise, if the AFHE ciphertexts are annihilating, then Eval queries SKE.Dec on random input. Note that this step represents an erroneous evaluation of the SKE.Dec circuit that in practice should result in an AFHE encryption of "rubish" when evaluated on annihilating ciphertexts. Before running RealExp or IdealExp, the challenger D queries SKE.Enc on keys according to a query specification $\mathcal{Q}_{\mathsf{Spec}}$. In the definition, we denote this set of queries to SKE.Enc as $\mathcal{Q}_{\mathsf{Chal}}$. The challenger is given a set of keys/messages $[\underline{K}_k]_{k=1}^{\rho}$ on which it makes the queries. Furthermore, we give $\mathcal{Q}_{\mathsf{Spec}}$ to A, as we only want to hide the keys and not the relations between the SKE generated ciphertexts. We register all queries to SKE due to the adversary's activity in $\mathcal{Q}_{\mathsf{A}}$. Finally, we say that AFHE is secure if, for all $\mathcal{Q}_{\mathsf{Spec}}$, and all sets of keys/messages $[\underline{K}_k]_{k=1}^{\rho}$ and annihilating tag specifications $\mathcal{K}$, querying SKE by any **PPT** adversary A on every query in $\mathcal{Q}_{\mathsf{Chal}}$ is equally (computationally close) probable in RealExp as in IdealExp. In other words, A getting real encryptions should not be capable of issuing queries to SKE that contradict the ideal model, and, in particular, are queries on keys encrypted under annihilating tags.

Note that, when secret keys are encrypted under annihilating tags, but it is possible to decrypt the keys, it is also possible to issue queries to SKE on those keys. In IdealExp, this possibility is reduced only to a lucky guess. Hence our definition already captures some notion of secrecy. It seems that a semantically secure AFHE would satisfy the secrecy requirement, but we do not know of any formal reduction.

An important caveat of our definition is that we use the ideal cipher as a black box and a circuit simultaneously. While such an implicit assumption is not a "clean," use of idealized objects in cryptography, we note that our work is not the first to exploit such duality. Notably, Gentry [Gen09, Theorem 4.4.2.] implicitly assumes a circuit representation of the random oracle that enables an adversary to query the oracle on encrypted data. Bitansky et al. [BCCT13, BCC+17] use an analogous assumption on generic groups for recursive proof composition.

Definition 12 (Ideal Cipher Annihilation Security). *Let* D *be a* **PPT** *algorithm that takes as input a set* $\mathcal{Q}_{\mathsf{Spec}} = [(\mathcal{S}_i, y_i)]_{i=1}^{\gamma}$, *where* $\mathcal{S}_i \subset [\rho]$ *and* $y_i \in [\rho]$, *and a set of keys* $[\underline{K}_k]_{k=1}^{\rho}$, *where* $\underline{K}_k \in \{0,1\}^{\ell_{\mathsf{sk}}}$. *The algorithm* D *outputs a set* $[\underline{\mathsf{ct}}_i]_{i=1}^{\gamma}$, *where* $\underline{\mathsf{ct}}_i \leftarrow \mathsf{SKE.Enc}([\underline{K}_k]_{k\in\mathcal{S}_i}, \underline{K}_{y_i})$ *for all* $i \in [\gamma]$. *We denote as* $\mathcal{Q}_{\mathsf{Chal}} = [([\underline{K}_k]_{k\in\mathcal{S}_i}, \underline{K}_{y_i}, \underline{\mathsf{ct}}_i)]_{i=1}^{\gamma}$ *the set of* D*'s queries to* SKE.Enc. *Finally, by* $\mathcal{Q}(\mathsf{RealExp}(.,.,.,.))$ *(resp.* $\mathcal{Q}(\mathsf{IdealExp}(.,.,.,.))$*) we denote the set of queries to* SKE *by* A *after running the real (resp. ideal) experiment.*

We say that AFHE *is secure, if for all* $\lambda \in \mathbb{N}$*, all* $\mathsf{poly}(\lambda)$ *size sets* $\mathcal{Q}_{\mathsf{Spec}}$ *as described above, all* $\kappa \leq \rho$*, all* $\underline{K}_k \xleftarrow{\$} \{0,1\}^{\ell_{\underline{\mathsf{sk}}}}$ *for* $k \in [\rho]$*, all* $\mathcal{K} \subseteq [\kappa] \times [\kappa]$*, all* $T \in [\gamma]$*, and all* $\mathbf{PPT}$ *adversaries* A*, we have that*

$$\left| \Pr\left[Q \in \mathcal{Q}_{\mathsf{A}} : \mathcal{Q}_{\mathsf{A}} \leftarrow \mathcal{Q}(\mathsf{IdealExp}(\lambda, [\underline{K}_k]_{k=1}^{\kappa}, [\underline{\mathsf{ct}}_i]_{i=1}^{\gamma}, \mathcal{K}, \mathcal{Q}_{\mathsf{Spec}}, T)) \right] \right.$$
$$\left. - \Pr\left[Q \in \mathcal{Q}_{\mathsf{A}} : \mathcal{Q}_{\mathsf{A}} \leftarrow \mathcal{Q}(\mathsf{RealExp}(\lambda, [\underline{K}_k]_{k=1}^{\kappa}, [\underline{\mathsf{ct}}_i]_{i=1}^{\gamma}, \mathcal{K}, \mathcal{Q}_{\mathsf{Spec}}, T)) \right] \right| \leq \mathsf{negl}(\lambda),$$

where the probability is over the random coins of AFHE.Setup *and* AFHE.Enc*, random choice of* $[\underline{K}_k]_{k=1}^{\rho}$*, and where the real experiment* RealExp *and ideal experiment* IdealExp *are defined as follows.*

$\mathsf{RealExp}(\lambda, [\underline{K}_k]_{k=1}^{\kappa}, [\underline{\mathsf{ct}}_i]_{i=1}^{\gamma}, \mathcal{K}, \mathcal{Q}_{\mathsf{Spec}}, T)$**:** *Takes as input a security parameter* λ*, a set of keys* $[\underline{K}_k]_{k=1}^{\kappa}$*, a set of ciphertexts* $[\underline{\mathsf{ct}}_i]_{i=1}^{\gamma}$*, a set of annihilating tags* $\mathcal{K} \subseteq [\kappa] \times [\kappa]$*, the query specification* $\mathcal{Q}_{\mathsf{Spec}}$ *and a target* $T \in [\gamma]$*.*

1. $(\overline{\mathsf{sk}}, \overline{\mathsf{ek}}) \leftarrow \mathsf{AFHE.Setup}(\lambda, \kappa, \mathcal{K})$*.*
2. $\overline{\mathsf{ct}}_k \leftarrow \mathsf{AFHE.Enc}(\overline{\mathsf{sk}}, \{k\}, \underline{K}_k)$ *for all* $k \in [\kappa]$*.*
3. $\mathsf{A}^{\mathsf{SKE}}([\underline{\mathsf{ct}}_i]_{i=1}^{\gamma}, \overline{\mathsf{ek}}, [\overline{\mathsf{ct}}_k]_{k=1}^{\kappa}, \mathcal{Q}_{\mathsf{Spec}}, T)$*.*

$\mathsf{IdealExp}(\lambda, [\underline{K}_k]_{k=1}^{\kappa}, [\underline{\mathsf{ct}}_i]_{i=1}^{\gamma}, \mathcal{K}, \mathcal{Q}_{\mathsf{Spec}}, T)$**:** *Takes as input a security parameter* λ*, a set of keys* $[\underline{K}_k]_{k=1}^{\kappa}$*, a set of ciphertexts* $[\underline{\mathsf{ct}}_i]_{i=1}^{\gamma}$*, a set of annihilating tags* $\mathcal{K} \subseteq [\kappa] \times [\kappa]$*, the query specification* $\mathcal{Q}_{\mathsf{Spec}}$ *and a target* $T \in [\gamma]$*.*

1. *Initiate the evaluation table* $\mathcal{E}$ *and set* $\overline{sk} \xleftarrow{\$} \{0,1\}^{\ell_{\overline{sk}}}$*.*
2. *For all* $k \in [\kappa]$
   - *choose* $\overline{\mathsf{ct}}_k \xleftarrow{\$} \{0,1\}^{\ell_{\overline{\mathsf{ct}}}}$*,*
   - *store the tuple* $(\overline{\mathsf{ct}}_k, \mathcal{T}_k, \underline{K}_k)$ *in* $\mathcal{E}$*, where* $\mathcal{T}_i \leftarrow \{k\}$*.*
3. $\mathsf{A}^{\mathsf{SKE},\mathsf{AFHE}}([\underline{\mathsf{ct}}_i]_{i=1}^{\gamma}, \overline{\mathsf{ek}}, [\overline{\mathsf{ct}}_k]_{k=1}^{\kappa}, \mathcal{Q}_{\mathsf{Spec}}, T)$*.*

The $\mathsf{SKE} = (\mathsf{Enc}, \mathsf{Dec})$ *is modeled as an ideal cipher. Furthermore, the oracle* AFHE.Enc*, upon receiving a query* $(\mathcal{T}, \underline{K})$*, chooses* $\overline{\mathsf{ct}} \xleftarrow{\$} \{0,1\}^{\ell_{\overline{\mathsf{ct}}}}$ *and stores* $(\overline{\mathsf{ct}}, \mathcal{T}, \underline{K})$ *into* $\mathcal{E}$*.* AFHE.Dec *on input* $\overline{sk}$ *and* $\overline{\mathsf{ct}}$ *returns* $\underline{K}$ *if* $\overline{\mathsf{ct}}$ *exists. And finally* AFHE.Eval *is an oracle that is defined as follows.*

$\mathsf{Eval}([\overline{\mathsf{ct}}_i]_{i=1}^{\tau}, \underline{\mathsf{ct}})$**:** *Takes as input ciphertexts* $[\overline{\mathsf{ct}}_i]_{i=1}^{\tau}$ *where* $\overline{\mathsf{ct}}_i \in \{0,1\}^{\ell_{\overline{\mathsf{ct}}}}$*, and a ciphertext* $\underline{\mathsf{ct}} \in \{0,1\}^{\ell_{\underline{\mathsf{ct}}}}$*.*

1. *If* $([\overline{\mathsf{ct}}_i]_{i=1}^{\tau}, \underline{\mathsf{ct}})$ *was queried earlier, then return the corresponding* $\overline{\mathsf{ct}}_{\mathsf{out}}$*.*
2. *Choose a uniform* $\overline{\mathsf{ct}}_{\mathsf{out}} \xleftarrow{\$} \{0,1\}^{\ell_{\overline{\mathsf{ct}}}}$*.*
3. *For each* $i \in [\tau]$ *find the tuple* $(\overline{\mathsf{ct}}_i, \mathcal{T}_i, \underline{K}_i)$ *in* $\mathcal{E}$*.*
4. *If* $\cup_{i=1}^{\tau} \mathcal{T}_i$ *contains annihilating tags according to* $\mathcal{K}$*, choose* $\underline{K}'_i \xleftarrow{\$} \{0,1\}^{\ell_{\underline{\mathsf{sk}}}}$ *for all* $i \in [\tau]$*. Otherwise set* $[\underline{K}'_i]_{i=1}^{\tau} \leftarrow [\underline{K}_i]_{i=1}^{\tau}$*.*
5. *Call* $\underline{K}_{\mathsf{out}} \leftarrow \mathsf{SKE.Dec}([\underline{K}'_i]_{i=1}^{\tau}, \underline{\mathsf{ct}})$*.*
6. *Set* $\mathcal{T}_{\mathsf{out}} = \cup_{i=1}^{\tau} \mathcal{T}_i$ *and store* $(\overline{\mathsf{ct}}_{out}, \mathcal{T}_{\mathsf{out}}, \underline{K}_{\mathsf{out}})$ *in the* $\mathcal{E}$*.*
7. *Store* $([\overline{\mathsf{ct}}_i]_{i=1}^{\tau}, \underline{\mathsf{ct}}, \overline{\mathsf{ct}}_{\mathsf{out}})$ *for a future call with the same inputs.*
8. *Return* $\overline{\mathsf{ct}}_{\mathsf{out}}$*.*

3.2 Candidate Annihilating (Leveled) FHE

In this section, we give our candidate AFHE in its basic version, i.e., without optimizations. Our candidate is based on a multikey FHE variant of $\mathsf{BGV}/\mathsf{BFV}$ type FHE schemes [Bra12, BV11, BGV12, FV12].

Construction 1 ((Basic) Candidate Annihilating FHE) *The candidate annihilating fully homomorphic encryption* $\mathsf{AFHE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *is as follows.*

$\mathsf{Setup}(\lambda, \kappa, \mathcal{K})$**:** *The algorithm takes as input the security parameter* λ*, a number of keys* $\kappa \in \mathsf{poly}(\lambda)$ *and a set indicating which keys are annihilating* $\mathcal{K} \subseteq [\kappa] \times [\kappa]$*.*

 # Select the Parameters:

1. *Select error distributions* $\mathcal{X}_{\mathsf{ek}}$*,* $\mathcal{X}_{\mathsf{ct}}$*,* $\mathcal{X}_{\mathsf{sk}}$*, the degree* $N \in \mathbb{N}$ *and a modulus* $q \in \mathbb{N}$ *that define the ring* $\mathcal{R}_q$*.*
2. *Select* $B \in \mathbb{N}$ *defining the gadget vector* $\mathbf{g}_{B,q}$*, and* $\ell = \lceil \log_B q \rceil$*.*

 # Generate Secret Keys and Relinearization Keys:

3. *For all* $k \in [\kappa]$ *do:*
   - 3.1. *Choose* $\mathbf{s}_k \xleftarrow{\$} \mathcal{X}_{\mathsf{sk}}^{n \cdot \kappa}$*.*
   - 3.2. *For all* $i \in [n]$ *and* $l \in [\ell]$*:*
     - 3.2.1 *Compute* $\mathsf{ek}_{1,k}[i,l] \xleftarrow{\$} \mathsf{GLWE}_{\mathcal{X}_{\mathsf{ek}},n,N,q}(\mathbf{s}_k, \mathbf{g}[l] \cdot \mathbf{s}_k[i])$*.*
   - 3.3. *For all all* $i, i' \in [n]$ *and* $l \in [\ell]$*:*
     - 3.3.1 *Compute* $\mathsf{ek}_{2,k}[i,i',l] \xleftarrow{\$} \mathsf{GLWE}_{\mathcal{X}_{\mathsf{ek}},n,N,q}(\mathbf{s}_k, \mathbf{g}[l] \cdot \mathbf{s}_k[i] \cdot \mathbf{s}_k[i'])$*.*

 # Generate Multikey Relinearization Keys:

4. *Define the non-annihilating tags* $\overline{\mathcal{K}} = ([\kappa] \times [\kappa]) \setminus \mathcal{K}$*.*
5. *For all* $(k,j) \in \overline{\mathcal{K}}$ *do:*
   - 5.1. *Set* $\mathsf{mk}_{k,j}[i,i',l] \xleftarrow{\$} \mathsf{GLWE}_{\mathcal{X}_{\mathsf{ek}},n,N,q}([\mathbf{s}_k, \mathbf{s}_j], \mathbf{g}[l] \cdot \mathbf{s}_k[i] \cdot \mathbf{s}_j[i'])$ *for all* $i, i' \in [n]$ *and* $l \in [\ell]$*.*

 # Output the Relinearization Keys and the Secret Keys:

6. *Return* $\overline{\mathsf{ek}} = ([\mathsf{ek}_{k,1}, \mathsf{ek}_{k,2}]_{k \in [\kappa]}, [\mathsf{mk}_{k,j}]_{(k,j) \in \overline{\mathcal{K}}})$ *and* $\overline{\mathsf{sk}} = [\mathbf{s}_k]_{k=1}^{\kappa}$*.*

$\mathsf{Enc}(\overline{\mathsf{sk}}, \mathcal{T}, \mathsf{msg})$**:** *The algorithm takes as input the secret key* $\overline{\mathsf{sk}} = [\mathbf{s}_k]_{k=1}^{\kappa} \in \mathcal{R}_q^{\kappa \cdot n}$*, and a message* $\mathsf{msg} = \mathfrak{m} \in \mathcal{R}_q$*.*

1. *Set* $\mathbf{c} \leftarrow \mathsf{GLWE}_{\mathcal{X}_{\mathsf{ct}},n,N,q}([\mathbf{s}_k]_{k \in \mathcal{T}}, \mathfrak{m})$*.*
2. *Output* $\overline{\mathsf{ct}} = (\mathbf{c}, \mathcal{T})$*.*

$\mathsf{Dec}(\overline{\mathsf{sk}}, \overline{\mathsf{ct}})$**:** *Takes as input the secret key* $\overline{\mathsf{sk}} = [\mathbf{s}_k]_{k=1}^{\kappa} \in \mathcal{R}_q^{\kappa \cdot n}$*, and a ciphertext* $\overline{\mathsf{ct}} = (\mathbf{c}, \mathcal{T})$*.*

1. *Run* $\overline{\mathsf{ct}}' \leftarrow \mathsf{Lift}(\overline{\mathsf{ct}}, [\kappa])$*, such that* $\overline{\mathsf{ct}}' = (\mathbf{c}, [\kappa])$*, where* $\mathbf{c} \in \mathcal{R}_q^{\kappa \cdot n + 1}$*.*
2. *Output* $\lfloor \langle \mathbf{c}, [1, [\mathbf{s}_k]_{k=1}^{\kappa}] \rangle \rceil$*, where* $\lfloor . \rceil$ *is the appropriate rounding function.*

$\mathsf{Eval}([\overline{\mathsf{ct}}_j]_{j=1}^{\eta}, \overline{\mathsf{ek}}, C)$**:** *The algorithm takes as input ciphertexts* $[\overline{\mathsf{ct}}_j]_{j=1}^{\eta}$*, a evaluation key* $\overline{\mathsf{ek}}$ *and a circuit* C*. The algorithm evaluates* $\overline{\mathsf{ct}}_{\mathsf{out}} \leftarrow C(\overline{\mathsf{ct}}_1, \ldots, \overline{\mathsf{ct}}_\eta)$*, by means of the following operations.*

$\mathsf{Lift}(\overline{\mathsf{ct}}, \mathcal{T})$**:** *The algorithm takes as input a ciphertext* $\overline{\mathsf{ct}} = (\mathbf{c}, \mathcal{T}_{\mathsf{ct}})$*, where* $\mathbf{c} = \mathsf{GLWE}_{\mathcal{X},n,N,q}([\mathbf{s}_k]_{k\in\mathcal{T}_{\mathsf{ct}}}, .)$*.*

1. *Parse* $\mathbf{c} = [\mathfrak{b}, [\mathbf{a}_k]_{k\in\mathcal{T}_{\mathsf{ct}}}]^\top$*, where* $\mathbf{a}_k \in \mathcal{R}_q^n$*.*

 # Set Appropriate Entries of the Ciphertext to 0:

2. *We set* $\hat{\mathbf{c}} = [\hat{\mathfrak{b}}, [\hat{\mathbf{a}}_k]_{k\in\mathcal{T}\cup\mathcal{T}_{\mathsf{ct}}}]$*, such that* $\hat{\mathfrak{b}} = \mathfrak{b}$*, and*
   - *for all* $k \in \mathcal{T} \setminus \mathcal{T}_{\mathsf{ct}}$ *we have* $\hat{\mathbf{a}}_k = \mathbf{0} \in \mathcal{R}_q^n$*, and*
   - *for all* $k \in \mathcal{T}_{\mathsf{ct}}$*, set* $\hat{\mathbf{a}}_k = \mathbf{a}_k \in \mathcal{R}_q^n$*.*
3. *Return* $\overline{\mathsf{ct}}_{\mathsf{out}} = (\hat{\mathbf{c}}, \mathcal{T} \cup \mathcal{T}_{\mathsf{ct}})$*.*

$\mathsf{Add}(\overline{\mathsf{ct}}, \mathfrak{a})$**:** *Takes as input* $\overline{\mathsf{ct}} = (\mathbf{c}, \mathcal{T})$*, where* $\mathbf{c} \in \mathcal{R}_q^{d\cdot n+1}$ *and* $\mathfrak{a} \in \mathcal{R}_q$*.*

1. *Return* $\overline{\mathsf{ct}}_{\mathsf{out}} = (\mathbf{c} + [\mathfrak{a}, \mathbf{0}], \mathcal{T})$*.*

$\mathsf{Add}(\overline{\mathsf{ct}}_1, \overline{\mathsf{ct}}_2)$**:** *Takes as input ciphertexts* $\overline{\mathsf{ct}}_1 = (\mathbf{c}_1, \mathcal{T}_1)$*, and* $\overline{\mathsf{ct}}_2 = (\mathbf{c}_2, \mathcal{T}_2)$*.*

1. *Run* $\overline{\mathsf{ct}}_1' \leftarrow \mathsf{Lift}(\overline{\mathsf{ct}}_1, \mathcal{T}_2)$ *and* $\overline{\mathsf{ct}}_2' \leftarrow \mathsf{Lift}(\overline{\mathsf{ct}}_2, \mathcal{T}_1)$*.*
2. *Parse* $\overline{\mathsf{ct}}_1' = (\mathbf{c}_1', \mathcal{T}_1 \cup \mathcal{T}_2)$ *and* $\overline{\mathsf{ct}}_2' = (\mathbf{c}_2', \mathcal{T}_1 \cup \mathcal{T}_2)$*.*
3. *Output* $\overline{\mathsf{ct}}_{\mathsf{out}} \leftarrow (\mathbf{c}_1' + \mathbf{c}_2', \mathcal{T}_1 \cup \mathcal{T}_2)$*.*

$\mathsf{Mul}(\overline{\mathsf{ct}}, \mathfrak{a})$**:** *Takes as input* $\overline{\mathsf{ct}} = (\mathbf{c}, \mathcal{T})$*, where* $\mathbf{c} \in \mathcal{R}_q^{d\cdot n+1}$ *and* $\mathfrak{a} \in \mathcal{R}_q$*.*

1. *Return* $\overline{\mathsf{ct}}_{\mathsf{out}} \leftarrow (\mathbf{c} \cdot \mathfrak{a}, \mathcal{T})$*.*

$\mathsf{Mul}(\overline{\mathsf{ct}}_1, \overline{\mathsf{ct}}_2, \overline{\mathsf{ek}})$**:** *Takes as input* $\overline{\mathsf{ct}}_1 = (\mathbf{c}_1, \mathcal{T}_1)$*,* $\overline{\mathsf{ct}}_2 = (\mathbf{c}_2, \mathcal{T}_2)$ *and the relinearization keys* $\overline{\mathsf{ek}} = ([\mathsf{ek}_{1,k}]_{k=1}^{\kappa}, [\mathsf{ek}_{2,k}]_{k=1}^{\kappa}, [\mathsf{mk}_{k,j}]_{k,j\in\overline{\mathcal{K}}})$*.*

 # Lift Both Ciphertexts:

1. *Set* $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2$*.*
2. *Run* $\hat{\mathsf{ct}} \leftarrow \mathsf{Lift}(\overline{\mathsf{ct}}_1, \mathcal{T}_2)$ *and* $\hat{\mathsf{ct}}' \leftarrow \mathsf{Lift}(\overline{\mathsf{ct}}_2, \mathcal{T}_1)$*.*
3. *Parse* $\hat{\mathsf{ct}} = (\hat{\mathbf{c}}, \mathcal{T})$ *and* $\hat{\mathsf{ct}}' = (\hat{\mathbf{c}}', \mathcal{T})$
4. *Parse* $\hat{\mathbf{c}} = [\hat{\mathfrak{b}}, [\hat{\mathbf{a}}_k]_{k\in\mathcal{T}}]$ *and* $\hat{\mathbf{c}}' = [\hat{\mathfrak{b}}', [\hat{\mathbf{a}}_k']_{k\in\mathcal{T}}]$*.*

 # Lift the Relinearization Keys:

5. *For all* $k \in \mathcal{T}$*,* $i \in [n]$ *and* $l \in [\ell]$ *set* $\hat{\mathsf{ek}}_{1,k}[i,l] \leftarrow \mathsf{Lift}(\mathsf{ek}_{1,k}[i,l], \mathcal{T})$*.*
6. *For all* $k \in \mathcal{T}$*,* $(i,i') \in [n]$ *and* $l \in [\ell]$ *compute* $\hat{\mathsf{ek}}_{2,k}[i,i',l] \leftarrow \mathsf{Lift}(\mathsf{ek}_{2,k}[i,i',l], \mathcal{T})$*.*
7. *For all* $(k,j) \in \overline{\mathcal{K}}$ *all* $(i,i') \in [n]$ *and* $l \in [\ell]$ *compute* $\hat{\mathsf{mk}}_{k,j}[i,i',l] \leftarrow \mathsf{Lift}(\mathsf{mk}_{k,j}[i,i',l], \mathcal{T})$*.*

 # Relinearize the (Extended) Ciphertext:

8. *Compute*

$$\mathbf{c}_{\mathsf{out}} = [\hat{\mathfrak{b}} \cdot \hat{\mathfrak{b}}', \mathbf{0}] + \sum_{k\in\mathcal{T}} \sum_{i=1}^{n} \mathsf{G}_{B,q}^{-1}(\hat{\mathfrak{b}}' \cdot \hat{\mathbf{a}}_k[i] + \hat{\mathbf{a}}_k'[i] \cdot \hat{\mathfrak{b}}) \cdot \hat{\mathsf{ek}}_{1,k}[i,*]^\top \quad (1)$$

$$+ \sum_{k\in\mathcal{T}} \sum_{\substack{i=1,\\ i'=1}}^{n} \mathsf{G}_{B,q}^{-1}(\hat{\mathbf{a}}_k[i] \cdot \hat{\mathbf{a}}_k'[i']) \cdot \hat{\mathsf{ek}}_{2,k}[i,i',*]^\top \quad (2)$$

$$+ \sum_{\substack{k,j\in\mathcal{T},\\ k\neq j}} \sum_{\substack{i=1,\\ i'=1}}^{n} \mathsf{G}_{B,q}^{-1}(\hat{\mathbf{a}}_k[i] \cdot \hat{\mathbf{a}}_j'[i']) \cdot \hat{\mathsf{mk}}_{k,j}[i,i',*]^\top \quad (3)$$

 # Return the Relinearized Multikey Ciphertext:

9. *Output* $\overline{\mathsf{ct}}_{\mathsf{out}} = (\mathbf{c}_{\mathsf{out}}, \mathcal{T})$*.*

Remark 2 (On the full construction). Above, we describe only the basic cryptosystem, and we omit many optimizations [GHS12, CS16]. To encode messages, we may use both the most significant (MSB) and least significant bit (LSB) encodings. It is worth noting that our scheme is a special case of the BGV/BFV-type cryptosystems, and security analysis and optimizations for those systems apply in our case as well. We note that the basic scheme requires circular security of GLWE samples to show semantic security. For completeness, we discuss the full scheme and recall the MSB and LSB encodings in Appendix C, where we also give (a sketch of) the proof for semantic security without the circularity assumption.

Below we sketch the correctness of our candidate, omitting technicalities like noise analysis. We give the full noise analysis of the scheme in Appendix B. The correctness of linear operations follows from linear homomorphism of GLWE samples. Hence we focus here on lifting and multiplication.

Theorem 1 (Correctness (Informal)). *The scheme given by Construction 1 is correct.*

Proof (Sketch). To argue correctness of lifting, let $\mathbf{c} = \mathsf{GLWE}_{\mathcal{X},n,N,q}([\mathbf{s}_k]_{k\in\mathcal{T}_{\mathsf{ct}}}, \mathfrak{m})$ be the input ciphertext. Note that $\hat{\mathbf{c}}$ is equivalent except we add $\mathbf{a}_k = \mathbf{0}$, for all $k \in \mathcal{T} \setminus \mathcal{T}_{\mathsf{ct}}$. Hence, we have $\langle \hat{\mathbf{c}}, [1, [\mathbf{s}_k]_{\mathcal{T}\cup\mathcal{T}_{\mathsf{ct}}}]\rangle = \langle \mathbf{c}, [1, [\mathbf{s}_k]_{\mathcal{T}_{\mathsf{ct}}}]\rangle$.

Showing correctness of multiplication between two ciphertexts is slightly more involved. Let us first recall how computing the Kronecker product of two ciphertexts works. Let $\hat{\mathbf{c}} = [\hat{\mathfrak{b}}, [\hat{\mathbf{a}}_k]_{k\in\mathcal{T}}]$ and $\hat{\mathbf{c}}' = [\hat{\mathfrak{b}}', [\hat{\mathbf{a}}'_k]_{k\in\mathcal{T}}]$. Let $\tilde{\mathbf{c}} = \hat{\mathbf{c}} \otimes \hat{\mathbf{c}}'$, be the extended ciphertext. We have that $\hat{\mathbf{c}}$ is a correct ciphertext with respect to $\tilde{\mathbf{s}} = \hat{\mathbf{s}} \otimes \hat{\mathbf{s}}$, where $\hat{\mathbf{s}} = [1, [\mathbf{s}_k]_{k\in\mathcal{T}}]$. To see this, note that from the mixed-product property of the Kronecker product we have $\langle \tilde{\mathbf{c}}, \tilde{\mathbf{s}}\rangle = \langle \hat{\mathbf{c}} \otimes \hat{\mathbf{c}}', \hat{\mathbf{s}} \otimes \hat{\mathbf{s}}\rangle = \langle \hat{\mathbf{c}}, \hat{\mathbf{s}}\rangle \cdot \langle \hat{\mathbf{c}}', \hat{\mathbf{s}}\rangle$.

Now consider the relinearization step. Note that we can rewrite line (1).

$$
\begin{aligned}
&[\hat{\mathfrak{b}} \cdot \hat{\mathfrak{b}}', \mathbf{0}] + \sum_{k\in\mathcal{T}} \sum_{i=1}^{n} \mathsf{G}^{-1}_{B,q}(\hat{\mathfrak{b}}' \cdot \hat{\mathbf{a}}_k[i] + \hat{\mathbf{a}}'_k[i] \cdot \hat{\mathfrak{b}}) \cdot \hat{\mathsf{ek}}_{1,k}[i, *]^{\top} \\
= \;& \mathsf{GLWE}_{\mathcal{X}_1,n,N,q}(\hat{\mathfrak{b}} \cdot \hat{\mathfrak{b}}' + \sum_{k\in\mathcal{T}} \sum_{i=1}^{n} \hat{\mathfrak{b}}' \cdot \hat{\mathbf{a}}_k[i] + \hat{\mathbf{a}}'_k[i] \cdot \hat{\mathfrak{b}} \cdot \mathfrak{s}_k) \\
= \;& \mathsf{GLWE}_{\mathcal{X}_1,n,N,q}(\hat{\mathfrak{b}} \cdot \hat{\mathfrak{b}}' + \hat{\mathfrak{b}}' \cdot \langle [\hat{\mathbf{a}}_k]_{k\in\mathcal{T}}, [\mathbf{s}_k]_{k\in\mathcal{T}}\rangle) + \hat{\mathfrak{b}} \cdot \langle [\hat{\mathbf{a}}'_k]_{k\in\mathcal{T}}, [\mathbf{s}_k]_{k\in\mathcal{T}}\rangle) \qquad (4)
\end{aligned}
$$

Then we have that the sum of the lines 2 and 3 of the relinearization step is equal to a $\mathsf{GLWE}_{\mathcal{X}_{2,3},n,N,q}$ sample with some error distribution $\mathcal{X}_{2,3}$ of the message:

$$
\begin{aligned}
&\sum_{k\in\mathcal{T}} \sum_{\substack{i=1,\\ i'=1}}^{n} \hat{\mathbf{a}}_k[i] \cdot \hat{\mathbf{a}}'_k[i'] \cdot \mathbf{s}_k[i] \cdot \mathbf{s}_k[i'] + \sum_{\substack{k,j\in\mathcal{T},\\ k\neq j}} \sum_{\substack{i=1,\\ i'=1}}^{n} \hat{\mathbf{a}}_k[i] \cdot \hat{\mathbf{a}}'_j[i'] \cdot \mathbf{s}_k[i] \cdot \mathbf{s}_k[i'] \qquad (5) \\
= \;& \sum_{k,j\in\mathcal{T}} \sum_{\substack{i=1,\\ i'=1}}^{n} \hat{\mathbf{a}}_k[i] \cdot \hat{\mathbf{a}}'_j[i'] \cdot \mathbf{s}_k[i] \cdot \mathbf{s}_k[i'] \\
= \;& \langle [\hat{\mathbf{a}}_k]_{k\in\mathcal{T}} \otimes [\hat{\mathbf{a}}_k]_{k\in\mathcal{T}}, [\hat{\mathbf{s}}_k]_{k\in\mathcal{T}} \otimes [\hat{\mathbf{s}}_k]_{k\in\mathcal{T}}\rangle \qquad (6)
\end{aligned}
$$

Finally, from the sum of 4 and 6 we have

$$\begin{aligned}
&\mathsf{GLWE}_{\mathcal{X}_{\mathsf{out}},n,N,q}(\hat{\mathfrak{b}}\cdot\hat{\mathfrak{b}}' + \hat{\mathfrak{b}}'\cdot\langle[\hat{\mathbf{a}}_k]_{k\in\mathcal{T}},[\mathbf{s}_k]_{k\in\mathcal{T}}\rangle) + \hat{\mathfrak{b}}\cdot\langle[\hat{\mathbf{a}}'_k]_{k\in\mathcal{T}},[\mathbf{s}_k]_{k\in\mathcal{T}}\rangle \\
&\qquad +\langle[\hat{\mathbf{a}}_k]_{k\in\mathcal{T}}\otimes[\hat{\mathbf{a}}_k]_{k\in\mathcal{T}},[\hat{\mathbf{s}}_k]_{k\in\mathcal{T}}\otimes[\hat{\mathbf{s}}_k]_{k\in\mathcal{T}}\rangle) \\
=\ &\mathsf{GLWE}_{\mathcal{X}_{\mathsf{out}},n,N,q}(\langle[\hat{\mathfrak{b}},[\hat{\mathbf{a}}_k]_{k\in\mathcal{T}}]\otimes[\hat{\mathfrak{b}}',[\hat{\mathbf{a}}_k]_{k\in\mathcal{T}}],[1,[\hat{\mathbf{s}}_k]_{k\in\mathcal{T}}]\otimes[1,[\hat{\mathbf{s}}_k]_{k\in\mathcal{T}}]\rangle) \\
=\ &\mathsf{GLWE}_{\mathcal{X}_{\mathsf{out}},n,N,q}(\langle\hat{\mathsf{ct}}\otimes\hat{\mathsf{ct}}',\hat{\mathsf{sk}}\otimes\hat{\mathsf{sk}}\rangle) \\
=\ &\mathsf{GLWE}_{\mathcal{X}_{\mathsf{out}},n,N,q}(\langle\hat{\mathsf{ct}},\hat{\mathsf{sk}}\rangle\cdot\langle\hat{\mathsf{ct}}',\hat{\mathsf{sk}}\rangle) \\
=\ &\mathsf{GLWE}_{\mathcal{X}_{\mathsf{out}},n,N,q}(\mathfrak{m}_1\cdot\mathfrak{m}_2)
\end{aligned}$$

3.3 Circular Insecure Version

We recall a generic compiler, due to Goyal, Koppula, and Waters [GKW17a], that compiles any (bit) encryption scheme to a circular insecure version.

Construction 2 (AFHE with an 1-Cycle Tester) *Let* $\mathsf{AFHE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *be an annihilating (leveled) fully homomorphic encryption scheme. Denote as* $\mathcal{S}$ *the secret key space and as* $\mathcal{M}$ *the message space induced by the execution of* $\mathsf{AFHE.Setup}$. *Let* $\mathsf{lockObf} = (\mathsf{Obf}, \mathsf{Eval})$ *be a lockable obfuscation scheme. Let* F *be the function that takes as input* $(\overline{\mathsf{sk}}, \mathsf{lock}) \in \mathcal{S}\times\mathcal{M}$, *and returns* lock. *We define an annihilating (leveled) fully homomorphic encryption with a* 1*-cycle tester for the function* F, *as* $\mathsf{ctAFHE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec}, \mathsf{CycleTest})$, *where* Enc, Eval *and* Dec *are as in* AFHE, *but* $\mathsf{ctAFHE.Setup}$ *and* $\mathsf{ctAFHE.CycleTest}$ *are as follows.*

$\mathsf{ctAFHE.Setup}(\lambda,\kappa,\mathcal{K})$**:** *Takes as input the security parameter* λ, *the number of key* κ, *and the set of annihilating tags* $\mathcal{K}$.

1. *Run* $(\overline{\mathsf{ek}}', \overline{\mathsf{sk}}') \leftarrow \mathsf{AFHE.Setup}(\lambda,\kappa,\mathcal{K})$.
2. *Choose* $\mathsf{lock} \xleftarrow{\$} \mathcal{M}$.
3. *Set* $C = \mathsf{AFHE.Dec}(\overline{\mathsf{sk}}', .)$ *and run* $\tilde{C} \leftarrow \mathsf{lockObf.Obf}(\lambda, C, \mathsf{lock})$.
4. *Return* $(\overline{\mathsf{ek}}, \overline{\mathsf{sk}})$, *where* $\overline{\mathsf{ek}} = (\overline{\mathsf{ek}}', \tilde{C})$ *and* $\overline{\mathsf{sk}} = (\overline{\mathsf{sk}}', \mathsf{lock})$.

$\mathsf{ctAFHE.CycleTest}(\overline{\mathsf{ek}}, \overline{\mathsf{ct}})$**:** *Takes an evaluation key* $\overline{\mathsf{ek}}$ *and a ciphertext* $\overline{\mathsf{ct}}$.

1. *Parse* $\overline{\mathsf{ek}} = (\overline{\mathsf{ek}}', \tilde{C})$.
2. *Return* 1 *if* $\mathsf{lockObf.Eval}(\tilde{C}, \overline{\mathsf{ct}}) = 1$. *Otherwise return* 0.

Remark 3 (Instantiating the Lockable Obfuscation). Aside from distributional virtual black-box security, we require that the lockable obfuscation doesn't increase the adversary's homomorphic capabilities. We can use the $\mathsf{GGH15}$ [GGH15] encoding based lockable obfuscator from [CVW18] to instantiate $\mathsf{lockObf}$, and we conjecture such $\mathsf{lockObf}$ does not help to break the ideal cipher annihilating security of our candidate. The high-level intuition for our conjecture is that $\mathsf{GGH15}$ encodings are designed to compute bounded length circuits in NC^1, which is enough for correctness but seems to stand in the way to compute more complex circuits naturally (i.e., without resorting bootstrapping techniques). For completeness, we recall the full construction of the lockable obfuscator from [CVW18] and the proof that the transformed cryptosystem preserves semantic security in Appendix C. We devise a more in-depth discussion on the conjectured security to the same section.

3.4 On the Security of our Candidate.

Semantic security follows from the fact that the adversary obtains only GLWE samples. The proof is a standard hybrid argument as in [BV11], for completeness, we recall a sketch in Appendix C.

Let us discuss how our candidate "destroys" plaintexts on annihilating tags for multiplication. Note that for annihilating tags k and j, we don't get $\mathsf{mk}_{k,j}$. As we can see by inspecting step 3 of relinearization in Equation 5 of the correctness proof, without relinearizing the $\mathsf{mk}_{k,j}$ part, the ciphertext decrypts to $\mathfrak{m} + \mathfrak{e} - \sum_{k,j\in\mathcal{T},k\neq j} \sum_{i=1,i'=1}^{n} \hat{\mathbf{a}}_k[i]\cdot\hat{\mathbf{a}}'_j[i']\cdot\mathbf{s}_k[i]\cdot\mathbf{s}_k[i']$, where $\mathfrak{m}$ is the product of the inputs and $\mathfrak{e}$ the noise. In other words, the element $\mathfrak{d} = -\sum_{k,j\in\mathcal{T},k\neq j} \sum_{i=1,i'=1}^{n} \hat{\mathbf{a}}_k[i] \cdot \hat{\mathbf{a}}'_j[i']\cdot\mathbf{s}_k[i]\cdot\mathbf{s}_k[i'] \in \mathcal{R}_q$ is added to the plaintext where $\hat{\mathbf{a}}_k[i]$ and $\hat{\mathbf{a}}'_j[i']$ are drawn uniformly from $\mathcal{R}_q$. The observation that $\mathfrak{d}$ is over $\mathcal{R}_q$ is crucial because this means that the risk that the error term $\mathfrak{e}$ will absorb $\mathfrak{d}$ is small.

A potential attack vector is to relinearize the ciphertext while keeping the noise within a reasonable bound. In other words, compute multiplication without the relinearization keys. Such an attack would mean that the symmetric versions of the BGV/BFV schemes are naturally multikey, and we could reduce the large memory requirement in the full version of the scheme where we publish relinearization keys for each level of the circuit. Note that such an attack could also be useful for relinearizing non-multikey BGV/BFV without the keys encrypting the quadratic part of the secret keys. Another strategy is to compute without relinearization and treat the resulting ciphertext as a ciphertext with respect to an extended secret key. Such an attack does not seem to be a viable option, as the ciphertext size grows exponentially. Finally, observe that computing the relinearisation keys holding powers-of-B of $\mathbf{s}_k[i]\cdot\mathbf{s}_k[i']$ from the key material available requires at least to have correct multikey multiplication in the first place.

Intuitively, even small errors in homomorphically evaluating the decryption circuit of an ideal cipher candidate should significantly distort the result of the computation. As we remarked in Section 2, the SKE can be realized by dividing, bits of the key into chunks. In particular, we might instantiate the scheme for fan-in $\tau = 2$, as $\mathsf{BlockCipher}(\underline{K}_1||\underline{K}_2, \mathsf{msg}')$, where $\underline{K}_i \in \{0,1\}^{128}$, $\mathsf{msg}' = \mathbf{0}||\mathsf{msg}$ and $\mathsf{msg} \in \{0,1\}^{128}$, and BlockCipher is for example AES256. We also note that cascade constructions like $\mathsf{BlockCipher}(\underline{K}_1, \mathsf{BlockCipher}(\underline{K}_2, \mathsf{msg}))$, as used in [BHR12], seem to be viable options that allow us to instantiate BlockCipher with AES128. However, we recall that cascades like this do not implement the ideal cipher model as we defined in Section 2, due to meet in the middle attacks.

Finally, note that our candidate AFHE allows us to compute affine functions on annihilating tags. While this fact contradicts the intuitive requirement that no computation is allowed on annihilating tags, for our security Definition 12, the inability to multiply should be enough. However, it is essential to keep that in mind when instantiating SKE. In particular, one should be careful and avoid instantiations like $\mathsf{msg} \oplus \mathsf{KDF}(\underline{K}_1) \oplus \mathsf{KDF}(\underline{K}_2)$ used in some implementations of garbled circuits [LPS08], where KDF is a key derivation function.

4 Our Witness Encryption Scheme

This section gives our main witness encryption scheme from Yao's garbling technique and annihilating FHE. To reduce complexity, we first give definitions for two subroutines in Subsection 4.1. In Subsection 4.2, we show our witness encryption scheme. In Subsection 4.3, we prove the security of our scheme.

4.1 Subroutines

Below we define our procedures that encode and decode a formula. For this purpose, for a gate g, we define a table $\mathsf{g.encG}[v_1, \ldots, v_\tau]$, that stores ciphertexts and is indexed by symbols associated with all τ incoming wires to that gate.

Construction 3 (Gate Encoding and Decoding) *Let λ be a security parameter and Λ an alphabet. Let $\mathsf{SKE} = (\mathsf{Enc}, \mathsf{Dec})$ be a symmetric encryption scheme with key size $\ell_{\underline{\mathsf{sk}}} = \mathsf{poly}(\lambda)$ and ciphertext size $\ell_{\underline{\mathsf{ct}}} = \mathsf{poly}(\lambda)$. The gate encoding algorithm* EncodeGate *and homomorphic gate decoding algorithm* HomDecodeGate *are as follows.*

$\mathsf{EncodeGate}(\mathsf{varLabels}, \mathsf{g}, [\underline{K}^v_{\mathsf{out}}]_{v\in\Lambda})$**:** *Takes as input, an array of variable labels* varLabels*, the gate* g *and the output wire labels $[\underline{K}^v_{\mathsf{out}}]_{v\in\Lambda} \in \{0,1\}^{\ell_{\underline{\mathsf{sk}}}\cdot|\Lambda|}$.*

1. *Set $\tau \leftarrow \mathsf{g}.\mathit{fan\text{-}in}()$.*

 # Choose the Labels of Internal Wires:

2. *For each $v \in \Lambda$ and each $j \in [\tau]$:*
   - 2.1. *If $\mathsf{g.child}(j) = \mathsf{null}$, then set $\underline{K}^v_j \leftarrow \mathsf{varLabels}[\mathsf{g.wireVar}(j), v]$.*
   - 2.2. *Otherwise, choose $\underline{K}^v_j \xleftarrow{\$} \{0,1\}^{\ell_{\underline{\mathsf{sk}}}}$ uniformly random.*

 # Encode the Gate According to its Truth Table:

3. *For each tuple $(v_1, \ldots, v_\tau) \in \Lambda^\tau$*
   - 3.1. *Compute $\mathsf{g.encG}[v_1, \ldots, v_\tau] \leftarrow \mathsf{SKE.Enc}([\underline{K}^{v_1}_1, \ldots, \underline{K}^{v_\tau}_\tau], \underline{K}^{v_{\mathsf{out}}}_{\mathsf{out}})$, where $v_{\mathsf{out}} \leftarrow \mathsf{g}(v_1, \ldots, v_\tau)$.*

 # Execute the Procedure Recursively:

4. *For each $j \in [\tau]$,*
   - 4.1. *if $\mathsf{g.child}(j) \neq \mathsf{null}$, then run $\mathsf{EncodeGate}(\mathsf{g.child}(j), [\underline{K}^v_j]_{v\in\Lambda})$.*

$\mathsf{HomDecodeGate}(\mathsf{varLabels}, \overline{\mathsf{ek}}, \mathsf{g}, \mathsf{wit})$**:** *Takes as input an array of* AFHE *ciphertexts* varLabels *and an evaluation key $\overline{\mathsf{ek}}$, a gate* g *and a witness* wit*.*

1. *Set $\tau \leftarrow \mathsf{g}.\mathit{fan\text{-}in}()$.*
2. *Set the values $(v_1, \ldots, v_\tau)$ of incoming wires to gate* g *according to* wit*.*

 # Obtain the Incoming Wire Labels According to the Witness:

3. *For all $j \in [\tau]$,*
   - 3.1. *if $\mathsf{g.child}(j) = \mathsf{null}$, then $\overline{\mathsf{ct}}^{v_j}_j \leftarrow \mathsf{varLabels}[\mathsf{g.wireVar}(j), v_j]$.*
   - 3.2. *otherwise, run $\overline{\mathsf{ct}}^{v_j}_j \leftarrow \mathsf{HomDecodeGate}(\mathsf{varLabels}, \mathsf{g.child}(j), \mathsf{wit})$.*

 # Homomorphically Decrypt the Output Wire:

4. *Compute*

$$\overline{\mathsf{ct}}_{\mathsf{out}} \leftarrow \mathsf{AFHE.Eval}([\overline{\mathsf{ct}}^{v_i}_j]^\tau_{j=1}, \overline{\mathsf{ek}}, \mathsf{SKE.Dec}(\ldots, \mathsf{g.encG}[v_1, \ldots, v_\tau])).$$

5. *Return $\overline{\mathsf{ct}}_{\mathsf{out}}$.*

4.2 Construction of the Witness Encryption

In this section, we show our witness encryption construction. In a nutshell, we encode a formula using the subroutines from Section 4.1. However, we set the label representing the accepting symbol to a function of the AFHE secret key if the message is 1. If the message is 0, then we choose the label of the accepting symbol uniformly at random. Then we encrypt the variable labels using the AFHE with a cycle tester. We set the tags of the AFHE ciphertexts such that ciphertexts holding labels that represent the same variable but different values are annihilating.

Construction 4 (Our Witness Encryption Scheme) *Let λ be a security parameter. Let* $\mathsf{SKE} = (\mathsf{Enc}, \mathsf{Dec})$ *be a symmetric encryption scheme with key size* $\ell_{\underline{\mathsf{sk}}} = \mathsf{poly}(\lambda)$ *and ciphertext size* $\ell_{\underline{\mathsf{ct}}} = \mathsf{poly}(\lambda)$. *Let* $\mathsf{AFHE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec}, \mathsf{CycleTest})$ *be an annihilating (leveled) homomorphic encryption with a* 1*-cycle tester with respect to a function* F. *The witness encryption scheme* $\mathsf{WE} = (\mathsf{Enc}, \mathsf{Dec})$ *is as follows.*

$\mathsf{Enc}(\lambda, \mathsf{stmt}, \mathsf{msg})$**:** *The algorithm takes as input a security parameter λ, a statement* stmt *and a message* $\mathsf{msg} \in \{0,1\}^{\ell_{\mathsf{msg}}}$. *We assume that the relation $\mathcal{R}$ associated with* stmt *can be represented as a formula C of η variables, over the alphabet Λ and with* $\mathsf{g}_{\mathsf{out}}$ *being the root of the formula.*

 # Set up the Annihilating Fully Homomorphic Encryption:

1. *Set the number of all tags* $\kappa \leftarrow |\Lambda| \cdot \eta$.
2. *Define $\mathcal{K}$ such that for all $i \in [\eta]$ and all $j, j' \in |\Lambda|$ such that $j \neq j'$, we have the following*

$$((i-1) \cdot |\Lambda| + j, (i-1) \cdot |\Lambda| + j') \in \mathcal{K}$$

3. *Run* $(\overline{\mathsf{ek}}, \overline{\mathsf{sk}}) \leftarrow \mathsf{AFHE.Setup}(\lambda, \kappa, \mathcal{K})$.

 # Create a Basic Witness Encryption of $\mathsf{F}(\overline{\mathsf{sk}})$ *or a Random String:*

4. *If* $\mathsf{msg} = 1$ *set* $\underline{\mathsf{unlock}} \xleftarrow{\$} \mathsf{F}(\overline{\mathsf{sk}})$. *Otherwise, choose* $\underline{\mathsf{unlock}} \xleftarrow{\$} \{0,1\}^{\ell_{\underline{\mathsf{sk}}}}$.
5. *For each $v \in \Lambda$*
   5.1. *if v is accepting then set* $\underline{K}^v_{\mathsf{out}} \leftarrow \underline{\mathsf{unlock}}$,
   5.2. *otherwise choose* $\underline{K}^v_{\mathsf{out}} \xleftarrow{\$} \{0,1\}^{\ell_{\underline{\mathsf{sk}}}}$ *uniformly at random.*
6. *For all input variables $i \in [\eta]$, and all $v \in \Lambda$:*
   6.1. *Choose* $\underline{K}^v_i \leftarrow \{0,1\}^{\ell_{\underline{\mathsf{sk}}}}$, *and set* $\mathsf{varLabels}[i, v] \leftarrow \underline{K}^v_i$.
7. *Run* $\mathsf{EncodeGate}(g_{\mathsf{out}}, [\underline{K}^v_{\mathsf{out}}]_{v \in \Lambda}, \mathsf{varLabels})$.

 # Encrypt the Variable Labels:

8. *For all $v \in \Lambda$ and all variables $i \in [\eta]$ compute*

$$\mathsf{varLabels}[i, v] \leftarrow \mathsf{AFHE.Enc}(\overline{\mathsf{sk}}, \{(i-1) \cdot |\Lambda| + j\}, \mathsf{varLabels}[i, v]),$$

where $j \in [|\Lambda|]$ is such that $v = \Lambda[j]$.

 # Return the Ciphertext:

9. *Return* $\mathsf{ct} \leftarrow (\mathsf{g}_{\mathsf{out}}, \mathsf{varLabels}, \overline{\mathsf{ek}})$.

$\mathsf{Dec}(\mathsf{stmt}, \mathsf{wit}, \mathsf{ct})$**:** *The algorithm takes as input a statement* stmt*, a witness* wit *and a ciphertext* ct.

1. *Parse* $\mathsf{ct} = (\mathsf{g}_{\mathsf{out}}, \mathsf{varLabels}, \overline{\mathsf{ek}}, \tilde{P})$.

`# Homomorphically Decode the Basic Witness Encryption:`

2. *Compute* $\overline{\mathsf{ct}}_{\mathsf{unlock}} \leftarrow \mathsf{HomDecodeGate}(\mathsf{varLabels}, \mathsf{g}_{\mathsf{out}}, \mathsf{wit})$.

`# Run the Cycle Tester and Return the Result:`

3. *Run* $\mathsf{msg} \leftarrow \mathsf{AFHE.CycleTest}(\overline{\mathsf{ek}}, \overline{\mathsf{ct}}_{\mathsf{unlock}})$.
4. *Return* msg.

Remark 4 (Arbitrary Cycle Length). For simplicity, we described our witness encryption scheme for a 1-cycle tester. Extending the construction of an n-cycle tester is straightforward. It suffices to run n witness encryption in parallel. If $\mathsf{msg} = 1$, then unlock_i in the ith scheme is set to $\mathsf{F}(\overline{\mathsf{sk}}_{(i \mod n)+1})$.

Theorem 2 (Correctness). *Let* stmt *be a formula* C *over alphabet* Λ *and with fan-in at most* τ*. Let* $\mathsf{SKE} = (\mathsf{Enc}, \mathsf{Dec})$ *be a perfectly correct symmetric encryption scheme with key size* $\ell_{\underline{\mathsf{sk}}} = \mathsf{poly}(\lambda)$ *and ciphertext size* $\ell_{\underline{\mathsf{ct}}} = \mathsf{poly}(\lambda)$*. Let* $\mathsf{AFHE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec}, \mathsf{CycleTest})$ *be an annihilating (leveled) homomorphic encryption with a* 1*-cycle tester with respect to a function* F*. Then* WE *for* stmt *is correct.*

Furthermore, encryption works in time $\mathsf{poly}(\lambda, |C| \cdot |\Lambda|^\tau)$*, decryption works in time* $\mathsf{poly}(\lambda, |C|)$ *and the size of a ciphertext is* $|\mathsf{ct}| = \mathsf{poly}(\lambda, |C| \cdot |\Lambda|^\tau)$.

Proof. The decrypter is given as input a statement stmt and witness wit such that $\mathcal{R}(\mathsf{stmt}, \mathsf{wit}) = 1$ and a ciphertext ct. Let $(v_1, \ldots, v_\tau)$ be the values of the incoming wires to a gate g according to the witness wit. Let us denote as $\underline{\mathsf{ct}} = \mathsf{SKE.Enc}([\underline{K}_j^{v_j}]_{j=1}^{\tau}, \underline{K}_{\mathsf{out}}^{v_{\mathsf{out}}})$ a ciphertext corresponding to an evaluation of a gate. If the decrypter has $\overline{\mathsf{ct}}_i = \mathsf{AFHE.Enc}(\overline{\mathsf{sk}}, \mathcal{T}_i, \underline{K}_i^{v_i})$ for $i \in [\tau]$ where $\mathcal{T} = \sum_{i \in [\tau]} \mathcal{T}_i$ doesn't contain annihilating tags, then from the fact that $\underline{K}_{\mathsf{out}}^{v_{\mathsf{out}}} = \mathsf{SKE.Dec}([\underline{K}_j^{v_j}]_{j=1}^{\tau}, \underline{\mathsf{ct}})$ holds by the correctness of the SKE cryptosystem, we have that $\mathsf{AFHE.Enc}(\overline{\mathsf{sk}}, \mathcal{T}, \underline{K}_{\mathsf{out}}^{v_{\mathsf{out}}}) = \mathsf{AFHE.Eval}([\overline{\mathsf{c}}_i]_{i=1}^{\tau}, \overline{\mathsf{ek}}, \mathsf{SKE.Dec}(., \underline{\mathsf{ct}}))$ holds from correctness of the AFHE cryptosystem. Hence the decrypter obtains $\overline{\mathsf{ct}}_{\mathsf{out}} = \mathsf{AFHE.Enc}(\overline{\mathsf{sk}}, \mathcal{T}, \underline{K}_{\mathsf{out}}^{v_{\mathsf{out}}})$, which is an encryption of the label associated with $\mathsf{g}(v_1, \ldots, v_\tau) = v_{\mathsf{out}}$. In the case the decrypter has no access to all labels, it recursively walks the tree ending up with an input gate and then gradually decodes all necessary labels. The decrypter starts the recursion from $\overline{\mathsf{ct}_{\mathsf{unlock}}} = \mathsf{AFHE.Enc}(\overline{\mathsf{sk}}, \mathcal{T}_{\mathsf{wit}}, \underline{\mathsf{unlock}})$. Note that each input variable is assigned only a single value, and the ciphertexts of the corresponding keys are not annihilating. Finally, from correctness of cycle tester we have that $\mathsf{AFHE.CycleTest}(\overline{\mathsf{ek}}, \overline{\mathsf{ct}}_{\mathsf{unlock}}) = 1$, if $\mathsf{unlock} = \overline{sk}$ and $\perp$ otherwise.

For efficiency, the encrypter needs to invoke $\mathsf{SKE.Enc}$ $|\Lambda|^\tau$ times per gate and the decrypter needs to invoke $\mathsf{SKE.Dec}$ once per gate. Thus the execution times are $\mathsf{poly}(\lambda, |C| \cdot |\Lambda|^\tau)$ and $\mathsf{poly}(\lambda, |C|)$ respectively. The ciphertext ct contains labels of all gates in the formula where each gabled gate consists of $|\Lambda|^\tau$ ciphertexts, hence $|\mathsf{ct}| = \mathsf{poly}(\lambda, |C| \cdot |\Lambda|^\tau)$. We omit including the number of input labels separately, as their number is upper-bounded by the formula size.

4.3 Security Analysis

In this section, we show the following theorem.

Theorem 3 (Security). *Let* $\mathsf{stmt} \notin \mathcal{L}$. *Given that the* SKE *is modeled as an ideal cipher,* AFHE *is an Annihilating* FHE *scheme that is secure in the sense of Definition 12, then* WE *given by Construction 4 is a secure witness-encryption scheme.*

Proof. We show the theorem by a sequence of hybrid experiments. In a nutshell, we start with an experiment where we encrypt $\mathsf{msg} = 0$ and choose $\underline{\mathsf{unlock}}$ at random. We will show that, given $\mathsf{stmt} \notin \mathcal{L}$, querying the SKE oracle by A on a key/ciphertext tuple that would require SKE to return $\underline{\mathsf{unlock}}$ is infeasible. Since, SKE never returns $\underline{\mathsf{unlock}}$, we may as well set $\underline{\mathsf{unlock}}$ to $\mathsf{F}(\overline{\mathsf{sk}})$ thus encrypting $\mathsf{msg} = 1$. Formally the proof goes as follows.

Hybrid 0. This is a witness encryption scheme as in Construction 4 encrypting the message $\mathsf{msg} = 0$. Note that in this case, $\underline{\mathsf{unlock}}$ and all other labels are chosen uniformly at random.

Hybrid 1. This hybrid is identical to **Hybrid 0**, except that we abort the experiment if A queries SKE.Dec on $(\underline{K}_1, \ldots, \underline{K}_\tau, \underline{\mathsf{ct}}_{\mathsf{out}}) \in \{0,1\}^{\tau \cdot \ell_{\underline{\mathsf{sk}}} + \ell_{\underline{\mathsf{ct}}}}$ such that $(\underline{K}_1, \ldots, \underline{K}_\tau, \underline{\mathsf{ct}}_{\mathsf{out}}, \underline{\mathsf{unlock}})$ is in $\mathcal{Q}_{\mathsf{Chal}}$, or A queries SKE.Enc on $(\underline{K}_1, \ldots, \underline{K}_\tau, \underline{\mathsf{unlock}})$.

Claim. Given that $\mathsf{stmt} \notin \mathcal{L}$ and AFHE satisfies the security given by Definition 12, **Hybrid 1** is indistinguishable from **Hybrid 0**.

Proof. From Definition 12, we have that the probability of querying the SKE oracle on a particular input is close to the probability of querying it in a system, where the annihilating FHE is ideal. In particular, we threat **Hybrid** 1 as the RealExp from Definition 12. Below we show that in IdealExp, the probability that A issues the queries related to the lock value is infeasible given $\mathsf{stmt} \notin \mathcal{L}$. Thus, if issuing such a query would be feasible in RealExp, we could use such an adversary to break the security of AFHE.
Let us consider the IdealExp from the Definition 12. Assume that A queries SKE directly or due to the Eval oracle, on a tuple related to $\underline{\mathsf{unlock}}$. Let us call this query the unlock query. In IdealExp, we can build an extractor Ext that, upon receiving the unlock query, extracts a valid witness from the registered oracle calls, thus contradicting that $\mathsf{stmt} \notin \mathcal{L}$.
The extractor Ext initializes the oracles SKE and AFHE, and initiates $\mathcal{E}$. Then Ext computes all ciphertexts in the WE scheme as in **Hybrid** 0. Note that $\mathcal{Q}_{\mathsf{Spec}}$ represents the query specification and $\mathcal{Q}_{\mathsf{Chal}}$ the set of queries to SKE. Ext also sets up a table $\mathcal{Q}_{\mathsf{A}}$ to register queries to SKE due to A's activity. For the ciphertexts of the AFHE scheme, Ext chooses $\mathsf{varLabels}[i, v] \xleftarrow{\$} \{0,1\}^{\ell_{\overline{\mathsf{ct}}}}$ uniformly at random for all $i \in [\eta]$ and all $v \in \Lambda$. In particular, we have $\kappa = \eta \cdot |\Lambda|$ and for each $i \in [\eta]$ and $v \in \Lambda$ we denote $k = i \cdot |\Lambda| + j$, where $j \in [|\Lambda|]$ is such that $v = \Lambda[j]$ and we set the ciphertexts from the

experiment as $\overline{\mathsf{ct}}_k = \mathsf{varLabels}[i, v]$. For each $k \in [\kappa]$ the extractor stores each $(\overline{\mathsf{ct}}_k, \mathcal{K}_k, \underline{K}_k)$ in $\mathcal{E}$ as specified in $\mathsf{IdealExp}$.
We will first rule out collisions and lucky guesses for the oracles. Let q denote all queries that A and the challenger make to the SKE and AFHE oracles.

- In case there is a pair of queries to $\mathsf{SKE.Enc}$, which outputs the same $\underline{\mathsf{ct}}$, the extractor aborts and returns $\perp$. This event may happen with probability at most $\mathsf{q}^2/2^{\ell_{\underline{\mathsf{ct}}}}$.
- If there is a collision when querying the oracle AFHE and writing to $\mathcal{E}$, then the extractor aborts and returns $\perp$. In other words, if the $\mathsf{AFHE.Enc}$ or $\mathsf{AFHE.Eval}$ oracle try to store a $\overline{\mathsf{ct}}$ into $\mathcal{E}$, but $\mathcal{E}$ already contains $\overline{\mathsf{ct}}$. This event may happen with probability at most $\mathsf{q}^2/2^{\ell_{\overline{\mathsf{ct}}}}$.
- If A makes a query to $\mathsf{AFHE.Dec}$ that includes $\overline{\mathsf{sk}} \in \{0,1\}^{\ell_{\overline{sk}}}$, then Ext aborts. This event may happen with probability at most $\mathsf{q}/2^{\ell_{\overline{sk}}}$.
- If A makes a direct query to SKE or AFHE on an input x such that x contains a substring $\underline{K} \in \{0,1\}^{\ell_{\underline{\mathsf{sk}}}}$ and $\underline{K}$ can be found in $\mathcal{Q}_{\mathsf{Chal}}$, then Ext aborts. Since A does not get any information on the keys stored in $\mathcal{Q}_{\mathsf{Chal}}$, this event may happen with probability at most $\mathsf{q}^2/2^{\ell_{\underline{\mathsf{sk}}}}$. Note that we upperbound $\rho \cdot \mathsf{q} \leq \mathsf{q}^2$, where $\rho \in \mathbb{N}$ denotes the number of all SKE keys used in WE.

Now we have that, for all keys $\underline{K} \in \{0,1\}^{\ell_{\underline{\mathsf{sk}}}}$ that appear in the $\mathcal{Q}_{\mathsf{A}}$ table, there must be a corresponding entry in $\mathcal{E}$ since we ruled out the possibility of A making a direct query that includes $\underline{K}$ to SKE. Furthermore, all these values were established in the Eval oracle, as we ruled out the possibility of A to luckily guessing them when querying $\mathsf{AFHE.Enc}$ or obtaining them from $\mathsf{AFHE.Dec}$.
Now suppose that A queries Eval on $([\overline{\mathsf{ct}}_i^{\mathsf{in}}]_{i=1}^{\tau}, \underline{\mathsf{ct}}_{\mathsf{out}})$, and let $[\mathcal{T}_i^{\mathsf{in}}, \underline{K}_i^{\mathsf{in}}]_{i=1}^{\tau}$ be the tags and keys corresponding to the ciphertexts $[\overline{\mathsf{ct}}_i^{\mathsf{in}}]_{i=1}^{\tau}$. Denote $\mathcal{T}_{\mathsf{wit}} = \cup_{i=1}^{\tau} \mathcal{T}_i^{\mathsf{in}}$ and suppose $\mathcal{T}_{\mathsf{wit}}$ does not contain annihilating tags. Finally, suppose that the tuple $(\underline{K}_1^{\mathsf{in}}, \ldots, \underline{K}_{\tau}^{\mathsf{in}}, \underline{\mathsf{ct}}_{\mathsf{out}}, \underline{\mathsf{unlock}})$ is in the $\mathcal{Q}_{\mathsf{Chal}}$ table. In other words, the query requires the ideal cipher to return $\underline{\mathsf{unlock}}$. In this case, the extractor Ext builds a witness wit by setting its ith variable to $\mathsf{wit}[i] \leftarrow \Lambda[j]$, where j is such that $(i-1) \cdot |\Lambda| + j \in \mathcal{T}_{\mathsf{wit}}$. Note that due to the fact that all indexes in $\mathcal{T}_{\mathsf{wit}}$ are not annihilating, there is no other $j' \neq j$ such that $(i-1) \cdot |\Lambda| + j' \in \mathcal{T}_{\mathsf{wit}}$, and so each variable $i \in [\eta]$ in the formula is assigned a unique value.
Since we ruled out lucky guesses and collisions, for each $\underline{K}_i^{\mathsf{in}}$ where $i \in [\tau]$ there must be an entry $([\underline{K}_j]_{j=1}^{\tau}, \overline{\mathsf{ct}}, \underline{K}_i^{\mathsf{in}})$ in $\mathcal{Q}_{\mathsf{A}}$, unless $\overline{\mathsf{ct}}_i^{\mathsf{in}}$ is given to A as input. From correctness of the WE scheme, i.e., the way $\overline{\mathsf{ct}}_i^{\mathsf{in}}$ is constructed, we have that the values $(v_1, \ldots, v_{\tau}, v_{\mathsf{in}}) \in \Lambda$ associated with the keys $(\underline{K}_1, \ldots, \underline{K}_{\tau}, \underline{K}_i^{\mathsf{in}})$ satisfy $\mathsf{g}(v_1, \ldots, v_{\tau}) = v_{\mathsf{in}}$ if $\underline{K}_i^{\mathsf{in}}$ represents an internal wire of the formula. If $\underline{K}_i^{\mathsf{in}}$ represents an input wire, i.e.,$\overline{\mathsf{ct}}_i^{\mathsf{in}}$ is given to A as input, then the corresponding variable in wit is set to the value the key represents. We follow this reasoning recursively until we end up with input wires.
Consistency of the witness, i.e., that all input wires are assigned the same value, follows, again, from the fact that we ruled out random guesses. Note

that having an inconsistent assignment, we would end up the recursion with at least two keys that represent two input wires with the same variable but different value. Since each invocation of Eval includes the corresponding tags into the output set, at some point in the computation, A must have queried the Eval oracle on annihilating tags. Without loss of generality, assume it is the lock query. In this case, Eval chooses the keys at random, but it might have happened that the keys are equal to $\underline{K}_1^{\mathsf{in}}, \ldots, \underline{K}_\tau^{\mathsf{in}}$. However, we have already ruled out such an event.

Hybrid 2. This hybrid is identical to **Hybrid 2** except we encrypt the message $\mathsf{msg} = 1$. In particular the $\underline{\mathsf{unlock}}$ is set to $\mathsf{F}(\overline{sk})$.
Since in **Hybrid 2** we ruled out queries to the ideal cipher that would require the oracle to return $\underline{\mathsf{unlock}}$, from an adversaries perspective **Hybrid 2** is identical to **Hybrid 2**.

Finally, we have that **Hybrid 0** and **Hybrid** 2 are indistinguishable, given that AFHE satisfies Definition 12.

5 Conclusions and Future Directions

We believe that our novel design paradigm that departs the usual $i\mathcal{O}$ or MMap's based constructions might lead to a better understanding of witness encryption and perhaps even practical instantiations.

Crucially, our construction of AFHE is only conjectured to satisfy the needed security. However, it seems that a class of attacks on our candidate may lead to interesting techniques useful to optimize some existing schemes. Nevertheless, a natural open question is whether one can show our construction's security under possibly novel but falsifiable and instance independent assumptions.

An exciting part of our construction is that we exploit the fact that the annihilating fully homomorphic scheme is circularly insecure. To the best of our knowledge, our work shows the first instance that circular insecure schemes are useful building blocks. We hope that our work will motivate further study of circular insecure systems from a protocol designer's perspective, instead only from the theoretical perspective.

References

Agr19. Shweta Agrawal. Indistinguishability obfuscation without multilinear maps: New methods for bootstrapping and instantiation. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part I*, volume 11476 of *Lecture Notes in Computer Science*, pages 191–225. Springer, Heidelberg, May 2019.

AJL+19. Prabhanjan Ananth, Aayush Jain, Huijia Lin, Christian Matt, and Amit Sahai. Indistinguishability obfuscation without multilinear maps: New paradigms via low degree weak pseudorandomness and security amplification. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances*

in Cryptology – CRYPTO 2019, Part III, volume 11694 of *Lecture Notes in Computer Science*, pages 284–332. Springer, Heidelberg, August 2019.

AP16. Navid Alamati and Chris Peikert. Three's compromised too: Circular insecurity for any cycle length from (ring-)LWE. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 659–680. Springer, Heidelberg, August 2016.

BC12. Nir Bitansky and Alessandro Chiesa. Succinct arguments from multi-prover interactive proofs and their efficiency benefits. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 255–272. Springer, Heidelberg, August 2012.

BCC+17. Nir Bitansky, Ran Canetti, Alessandro Chiesa, Shafi Goldwasser, Huijia Lin, Aviad Rubinstein, and Eran Tromer. The hunting of the snark. *Journal of Cryptology*, 30(4):989–1066, Oct 2017.

BCCT13. Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th Annual ACM Symposium on Theory of Computing*, pages 111–120. ACM Press, June 2013.

BDGM20a. Zvika Brakerski, Nico Döttling, Sanjam Garg, and Giulio Malavolta. Candidate iO from homomorphic encryption schemes. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 79–109. Springer, Heidelberg, May 2020.

BDGM20b. Zvika Brakerski, Nico Döttling, Sanjam Garg, and Giulio Malavolta. Factoring and pairings are not necessary for io: Circular-secure lwe suffices. Cryptology ePrint Archive, Report 2020/1024, 2020. https://eprint.iacr.org/2020/1024.

BGI+17. Saikrishna Badrinarayanan, Sanjam Garg, Yuval Ishai, Amit Sahai, and Akshay Wadia. Two-message witness indistinguishability and secure computation in the plain model from new assumptions. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017, Part III*, volume 10626 of *Lecture Notes in Computer Science*, pages 275–303. Springer, Heidelberg, December 2017.

BGV12. Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, *ITCS 2012: 3rd Innovations in Theoretical Computer Science*, pages 309–325. Association for Computing Machinery, January 2012.

BHJ+19. Boaz Barak, Samuel B. Hopkins, Aayush Jain, Pravesh Kothari, and Amit Sahai. Sum-of-squares meets program obfuscation, revisited. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part I*, volume 11476 of *Lecture Notes in Computer Science*, pages 226–250. Springer, Heidelberg, May 2019.

BHR12. Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 2012: 19th Conference on Computer and Communications Security*, pages 784–796. ACM Press, October 2012.

BIJ+20. James Bartusek, Yuval Ishai, Aayush Jain, Fermi Ma, Amit Sahai, and Mark Zhandry. Affine determinant programs: A framework for obfuscation

and witness encryption. In Thomas Vidick, editor, *ITCS 2020: 11th Innovations in Theoretical Computer Science Conference*, volume 151, pages 82:1–82:39. LIPIcs, January 2020.

BIOW20. Ohad Barta, Yuval Ishai, Rafail Ostrovsky, and David J. Wu. On succinct arguments and witness encryption from groups. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part I*, volume 12170 of *Lecture Notes in Computer Science*, pages 776–806. Springer, Heidelberg, August 2020.

BISW17. Dan Boneh, Yuval Ishai, Amit Sahai, and David J. Wu. Lattice-based SNARGs and their application to more efficient obfuscation. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part III*, volume 10212 of *Lecture Notes in Computer Science*, pages 247–277. Springer, Heidelberg, April / May 2017.

BISW18. Dan Boneh, Yuval Ishai, Amit Sahai, and David J. Wu. Quasi-optimal SNARGs via linear multi-prover interactive proofs. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part III*, volume 10822 of *Lecture Notes in Computer Science*, pages 222–255. Springer, Heidelberg, April / May 2018.

BJK+18. Zvika Brakerski, Aayush Jain, Ilan Komargodski, Alain Passelègue, and Daniel Wichs. Non-trivial witness encryption and null-iO from standard assumptions. In Dario Catalano and Roberto De Prisco, editors, *SCN 18: 11th International Conference on Security in Communication Networks*, volume 11035 of *Lecture Notes in Computer Science*, pages 425–441. Springer, Heidelberg, September 2018.

BKP19. Nir Bitansky, Dakshita Khurana, and Omer Paneth. Weak zero-knowledge beyond the black-box barrier. In Moses Charikar and Edith Cohen, editors, *51st Annual ACM Symposium on Theory of Computing*, pages 1091–1102. ACM Press, June 2019.

BP15. Nir Bitansky and Omer Paneth. ZAPs and non-interactive witness indistinguishability from indistinguishability obfuscation. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015: 12th Theory of Cryptography Conference, Part II*, volume 9015 of *Lecture Notes in Computer Science*, pages 401–427. Springer, Heidelberg, March 2015.

BP16. Zvika Brakerski and Renen Perlman. Lattice-based fully dynamic multikey FHE with short ciphertexts. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 190–213. Springer, Heidelberg, August 2016.

Bra12. Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 868–886. Springer, Heidelberg, August 2012.

BSW12. Dan Boneh, Gil Segev, and Brent Waters. Targeted malleability: homomorphic encryption for restricted computations. In Shafi Goldwasser, editor, *ITCS 2012: 3rd Innovations in Theoretical Computer Science*, pages 350–366. Association for Computing Machinery, January 2012.

BV11. Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In Rafail Ostrovsky, editor, *52nd*

Annual Symposium on Foundations of Computer Science, pages 97–106. IEEE Computer Society Press, October 2011.

CCS19. Hao Chen, Ilaria Chillotti, and Yongsoo Song. Multi-key homomorphic encryption from TFHE. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019, Part II*, volume 11922 of *Lecture Notes in Computer Science*, pages 446–472. Springer, Heidelberg, December 2019.

CDKS19. Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019: 26th Conference on Computer and Communications Security*, pages 395–412. ACM Press, November 2019.

CL01. Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In Birgit Pfitzmann, editor, *Advances in Cryptology – EUROCRYPT 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 93–118. Springer, Heidelberg, May 2001.

CM15. Michael Clear and Ciaran McGoldrick. Multi-identity and multi-key leveled FHE from learning with errors. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 630–656. Springer, Heidelberg, August 2015.

CPS08. Jean-Sébastien Coron, Jacques Patarin, and Yannick Seurin. The random oracle model and the ideal cipher model are equivalent. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 1–20. Springer, Heidelberg, August 2008.

CS16. Ana Costache and Nigel P. Smart. Which ring based somewhat homomorphic encryption scheme is best? In Kazue Sako, editor, *Topics in Cryptology – CT-RSA 2016*, volume 9610 of *Lecture Notes in Computer Science*, pages 325–340. Springer, Heidelberg, February / March 2016.

CVW18. Yilei Chen, Vinod Vaikuntanathan, and Hoeteck Wee. GGH15 beyond permutation branching programs: Proofs, attacks, and candidates. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 577–607. Springer, Heidelberg, August 2018.

FNV17. Antonio Faonio, Jesper Buus Nielsen, and Daniele Venturi. Predictable arguments of knowledge. In Serge Fehr, editor, *PKC 2017: 20th International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 10174 of *Lecture Notes in Computer Science*, pages 121–150. Springer, Heidelberg, March 2017.

FV12. Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012. http://eprint.iacr.org/2012/144.

Gen09. Craig Gentry. *A Fully Homomorphic Encryption Scheme*. PhD thesis, Stanford, CA, USA, 2009.

GGH+13. Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th Annual Symposium on Founda-*

tions of Computer Science, pages 40–49. IEEE Computer Society Press, October 2013.

GGH15. Craig Gentry, Sergey Gorbunov, and Shai Halevi. Graph-induced multilinear maps from lattices. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015: 12th Theory of Cryptography Conference, Part II*, volume 9015 of *Lecture Notes in Computer Science*, pages 498–527. Springer, Heidelberg, March 2015.

GGHR14. Sanjam Garg, Craig Gentry, Shai Halevi, and Mariana Raykova. Two-round secure MPC from indistinguishability obfuscation. In Yehuda Lindell, editor, *TCC 2014: 11th Theory of Cryptography Conference*, volume 8349 of *Lecture Notes in Computer Science*, pages 74–94. Springer, Heidelberg, February 2014.

GGSW13. Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th Annual ACM Symposium on Theory of Computing*, pages 467–476. ACM Press, June 2013.

GHS12. Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 850–867. Springer, Heidelberg, August 2012.

GKP+13. Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. How to run turing machines on encrypted data. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 536–553. Springer, Heidelberg, August 2013.

GKW17a. Rishab Goyal, Venkata Koppula, and Brent Waters. Lockable obfuscation. In Chris Umans, editor, *58th Annual Symposium on Foundations of Computer Science*, pages 612–621. IEEE Computer Society Press, October 2017.

GKW17b. Rishab Goyal, Venkata Koppula, and Brent Waters. Separating semantic and circular security for symmetric-key bit encryption from the learning with errors assumption. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part II*, volume 10211 of *Lecture Notes in Computer Science*, pages 528–557. Springer, Heidelberg, April / May 2017.

GLS15. S. Dov Gordon, Feng-Hao Liu, and Elaine Shi. Constant-round MPC with fairness and guarantee of output delivery. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *Advances in Cryptology – CRYPTO 2015, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 63–82. Springer, Heidelberg, August 2015.

GLW14. Craig Gentry, Allison B. Lewko, and Brent Waters. Witness encryption from instance independent assumptions. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 426–443. Springer, Heidelberg, August 2014.

GP20. Romain Gay and Rafael Pass. Indistinguishability obfuscation from circular security. Cryptology ePrint Archive, Report 2020/1010, 2020. https://eprint.iacr.org/2020/1010.

GSW13. Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster,

attribute-based. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 75–92. Springer, Heidelberg, August 2013.

JLMS19. Aayush Jain, Huijia Lin, Christian Matt, and Amit Sahai. How to leverage hardness of constant-degree expanding polynomials overa $\mathbb{R}$ to build $i\mathcal{O}$. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part I*, volume 11476 of *Lecture Notes in Computer Science*, pages 251–281. Springer, Heidelberg, May 2019.

KMN+14. Ilan Komargodski, Tal Moran, Moni Naor, Rafael Pass, Alon Rosen, and Eylon Yogev. One-way functions and (im)perfect obfuscation. In *55th Annual Symposium on Foundations of Computer Science*, pages 374–383. IEEE Computer Society Press, October 2014.

KNY17. Ilan Komargodski, Moni Naor, and Eylon Yogev. Secret-sharing for NP. *Journal of Cryptology*, 30(2):444–469, April 2017.

KW16. Venkata Koppula and Brent Waters. Circular security separations for arbitrary length cycles from LWE. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part II*, volume 9815 of *Lecture Notes in Computer Science*, pages 681–700. Springer, Heidelberg, August 2016.

LJKW18. Jia Liu, Tibor Jager, Saqib A. Kakvi, and Bogdan Warinschi. How to build time-lock encryption. *Designs, Codes and Cryptography*, 86(11):2549–2586, Nov 2018.

LPS08. Yehuda Lindell, Benny Pinkas, and Nigel P. Smart. Implementing two-party computation efficiently with security against malicious adversaries. In Rafail Ostrovsky, Roberto De Prisco, and Ivan Visconti, editors, *SCN 08: 6th International Conference on Security in Communication Networks*, volume 5229 of *Lecture Notes in Computer Science*, pages 2–20. Springer, Heidelberg, September 2008.

LPST16. Huijia Lin, Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation with non-trivial efficiency. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016: 19th International Conference on Theory and Practice of Public Key Cryptography, Part II*, volume 9615 of *Lecture Notes in Computer Science*, pages 447–462. Springer, Heidelberg, March 2016.

LT17. Huijia Lin and Stefano Tessaro. Indistinguishability obfuscation from trilinear maps and block-wise local PRGs. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 630–660. Springer, Heidelberg, August 2017.

LTV12. Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In Howard J. Karloff and Toniann Pitassi, editors, *44th Annual ACM Symposium on Theory of Computing*, pages 1219–1234. ACM Press, May 2012.

MW16. Pratyay Mukherjee and Daniel Wichs. Two round multiparty computation via multi-key FHE. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 735–763. Springer, Heidelberg, May 2016.

PS16. Chris Peikert and Sina Shiehian. Multi-key FHE from LWE, revisited. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B: 14th Theory of Cryptography Conference, Part II*, volume 9986 of *Lecture Notes in Computer Science*, pages 217–238. Springer, Heidelberg, October / November 2016.

Reg05. Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th Annual ACM Symposium on Theory of Computing*, pages 84–93. ACM Press, May 2005.

Sha49. C. E. Shannon. Communication theory of secrecy systems. *The Bell System Technical Journal*, 28(4):656–715, Oct 1949.

WW20. Hoeteck Wee and Daniel Wichs. Candidate obfuscation via oblivious lwe sampling. Cryptology ePrint Archive, Report 2020/1042, 2020. https://eprint.iacr.org/2020/1042.

WZ17. Daniel Wichs and Giorgos Zirdelis. Obfuscating compute-and-compare programs under LWE. In Chris Umans, editor, *58th Annual Symposium on Foundations of Computer Science*, pages 600–611. IEEE Computer Society Press, October 2017.

Yao86. Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science*, pages 162–167. IEEE Computer Society Press, October 1986.

Zha16. Mark Zhandry. How to avoid obfuscation using witness PRFs. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A: 13th Theory of Cryptography Conference, Part II*, volume 9563 of *Lecture Notes in Computer Science*, pages 421–448. Springer, Heidelberg, January 2016.

Appendices

A Basic Witness Encryption Scheme from Yao's Garbling Technique

This section defines the basic witness encryption scheme for read-once formulas built only from Yao's garbled circuits. We can instantiate the construction from symmetric primitives. While the construction itself is not very useful, as it works only for read-once formulas, the analysis may serve as an additional sanity check for our main witness encryption candidate. The witness encryption uses gate encoding algorithms as given by Construction 3, but, since we do not use annihilating FHE, we need to modify the gate decoding algorithm.

Construction 5 (Gate Encoding and Decoding) *Let λ be a security parameter and Λ an alphabet. Let $\mathsf{SKE} = (\mathsf{Enc}, \mathsf{Dec})$ be a symmetric encryption scheme with key size $\ell_{\underline{\mathsf{sk}}} = \mathsf{poly}(\lambda)$ and ciphertext size $\ell_{\underline{\mathsf{ct}}} = \mathsf{poly}(\lambda)$. We define the gate encoding* EncodeGate *as in Construction 3. The gate decoding algorithm* DecodeGate *is as follows.*

$\mathsf{DecodeGate}(\mathsf{varLabels}, \mathsf{g}, \mathsf{wit})$**:** *The algorithm takes as input an array of variable labels* varLabels*, a gate* g *and the witness* wit*. Let $(v_1, \ldots, v_\tau)$, be the values of the input wires of* g *according to* wit*.*

1. *Set $\tau \leftarrow \mathsf{g}.\mathit{fan\text{-}in}()$.*
2. *Set the values $(v_1, \ldots, v_\tau)$ of incoming wires to gate* g *according to the witness* wit*.*

 # Obtain the Incoming Wire Labels According to the Witness:

3. *For all $j \in [\tau]$,*
   - 3.1. *if $\mathsf{g.child}(j) = \mathsf{null}$, then $\underline{K}_j^{v_j} \leftarrow \mathsf{varLabels}[\mathsf{g.wireVar}(j), v_j]$.*
   - 3.2. *otherwise, run $\underline{K}_j^{v_j} \leftarrow \mathsf{DecodeGate}(\mathsf{varLabels}, \mathsf{g.child}(j), \mathsf{wit})$.*

 # Decrypt the Output Wire:

4. *Compute $\underline{K}_{\mathsf{out}} \leftarrow \mathsf{SKE.Dec}(\underline{K}_1^{v_1}, \ldots, \underline{K}_\tau^{v_\tau}; \mathsf{g.encG}[v_1, \ldots, v_\tau])$.*
5. *Return $\underline{K}_{\mathsf{out}}$.*

Now we are ready to specify the basic witness encryption scheme.

Construction 6 (Basic Witness Encryption) *Let λ be a security parameter. Let $\mathsf{SKE} = (\mathsf{Enc}, \mathsf{Dec})$ be a symmetric encryption scheme with key size $\ell_{\underline{\mathsf{sk}}} = \mathsf{poly}(\lambda)$ and ciphertext size $\ell_{\underline{\mathsf{ct}}} = \mathsf{poly}(\lambda)$. The witness encryption scheme $\mathsf{WE} = (\mathsf{Enc}, \mathsf{Dec})$ is as follows.*

$\mathsf{Enc}(\lambda, \mathsf{stmt}, \mathsf{msg})$**:** *The algorithm takes as input a security parameter λ, statement* stmt *and message* msg*. We assume that the relation $\mathcal{R}$ associated with* stmt *can be represented as a formula over the alphabet Λ of η variables and with $\mathsf{g}_{\mathsf{out}}$ being the root of the formula.*

 # Set Labels for the Output Wire of the Formula:

1. *For each $v \in \Lambda$*

1.1. if v is accepting then set $\underline{K}^v_{\mathsf{out}} \leftarrow \mathsf{msg}$,

1.2. otherwise choose $\underline{K}^v_{\mathsf{out}} \xleftarrow{\$} \{0,1\}^{\ell_{\underline{\mathsf{sk}}}}$ *uniformly at random.*

 # Choose the Labels for Input Variables:

2. For all input variables $i \in [\eta]$*, and all* $v \in \Lambda$*:*

2.1. Choose $\underline{K}^v_i \leftarrow \{0,1\}^{\ell_{\underline{\mathsf{sk}}}}$*, and set* $\mathsf{varLabels}[i, v] \leftarrow \underline{K}^v_i$.

 # Run Gate Encoding Starting from the Output Gate:

3. Run $\mathsf{EncodeGate}(g_{\mathsf{out}}, [\underline{K}^v_{\mathsf{out}}]_{v\in\Lambda}, \mathsf{varLabels})$.

 # Return the Encoded Formula and the Variable Labels:

4. Return $\mathsf{ct} \leftarrow (\mathsf{g}_{\mathsf{out}}, \mathsf{varLabels})$.

$\mathsf{Dec}(\mathsf{stmt}, \mathsf{wit}, \mathsf{ct})$**:** *The algorithm takes as input the statement* stmt*, witness* wit*, and ciphertext* ct*. Remind that the ciphertext* ct *is parsed as a formula, representing the relation* $\mathcal{R}$ *associated with* stmt*, with* $\mathsf{g}_{\mathsf{out}}$ *be the root of that formula.*

 # Run Gate Decoding Starting from the Output Gate:

1. Compute $\mathsf{msg} \leftarrow \mathsf{DecodeGate}(\mathsf{g}_{\mathsf{out}}, \mathsf{wit})$.

2. Return msg.

Theorem 4 (Correctness). *Let* stmt *be a formula C over alphabet Λ and with fan-in τ. If* $\mathsf{SKE} = (\mathsf{Enc}, \mathsf{Dec})$ *is a perfectly correct encryption scheme parameterized by* $\ell_{\underline{\mathsf{sk}}}$ *and* $\ell_{\underline{\mathsf{ct}}}$*, then* WE *for* stmt *is perfectly correct. Furthermore, for every security parameter* λ*, every message* $\mathsf{msg} \in \{0,1\}^{\ell_{\underline{\mathsf{sk}}}}$*,* $\mathsf{ct} \leftarrow \mathsf{WE.Enc}(\lambda, \mathsf{stmt}, \mathsf{msg})$ *works in time* $\mathsf{poly}(\lambda) \cdot |C| \cdot |\Lambda|^\tau$*,* $\mathsf{WE.Dec}(\mathsf{stmt}, \mathsf{wit}, \mathsf{ct})$ *works in time* $\mathsf{poly}(\lambda) \cdot |C|$ *and* $|\mathsf{ct}| = \ell_{\underline{\mathsf{ct}}} \cdot |C| \cdot |\Lambda|^\tau$.

Proof. The decrypter is given as input a statement stmt and witness wit such that $\mathcal{R}(\mathsf{stmt}, \mathsf{wit}) = 1$ and a ciphertext ct. Let $(v_1, \ldots, v_\tau)$ be the values of the incoming wires to a gate g according to the witness wit. Assume the decrypter has all the labels $(\underline{K}^{v_1}_1, \ldots, \underline{K}^{v_\tau}_\tau)$ corresponding to $(v_1, \ldots, v_\tau)$ gate g. Then the following

$$\underline{K}^{v_{\mathsf{out}}}_{\mathsf{out}} = \mathsf{SKE.Dec}(\underline{K}^{v_1}_1, \ldots, \underline{K}^{v_\tau}_\tau, \mathsf{SKE.Enc}(\underline{K}^{v_1}_1, \ldots, \underline{K}^{v_\tau}_\tau, \underline{K}^{v_{\mathsf{out}}}_{\mathsf{out}}))$$

holds by the correctness of the SKE cryptosystem. Hence the decrypter obtains $\underline{K}^{v_{\mathsf{out}}}_{\mathsf{out}}$, which is the label associated with $\mathsf{g}(v_1, \ldots, v_\tau) = v_{\mathsf{out}}$. In the case the decrypter has no access to all labels, it recursively walks the tree ending up with an input gate and then gradually decodes all necessary labels. Note that the recursion must end, since the ciphertext of the witness encryption includes all labels for all input wires.

For efficiency, it is easy to see that the encrypter needs to invoke $\mathsf{SKE.Enc}$ $|\Lambda|^\tau$ times per gate in the circuit and the decrypter needs to invoke SKE.Dec once per gate, thus the execution times are $\mathsf{poly}(\lambda) \cdot |C| \cdot |\Lambda|^\tau$ and $\mathsf{poly}(\lambda) \cdot |C|$ respectively. Furthermore, the ciphertext ct contains encodings of all gates in the circuit where each gate encoding consists of $|\Lambda|^\tau$ ciphertexts, hence $|\mathsf{ct}| = \ell_{\underline{\mathsf{ct}}} \cdot |C| \cdot |\Lambda|^\tau$. Note that we omit including the input labels, size these are upper-bounded by the circuit size.

A.1 Security Analysis

First, we recall the definition of extractable witness encryption introduced by Goldwasser et al. [GKP+13].

Definition 13 (Extractable Security). *A witness encryption scheme for language* $\mathcal{L} \in \mathsf{NP}$ *is extractable secure if for all* **PPT** *adversaries* A*, and all polynomials* q *there exists a polynomial-time extractor* Ext *and a polynomial* p*, such that for all auxiliary inputs* aux *and for all* $\mathsf{stmt} \in \{0,1\}^*$*, the following holds:*

$$\Pr\left[\mathsf{A}(\mathsf{stmt},\mathsf{ct},\mathsf{aux}) = \mathsf{msg} : \begin{array}{l}\mathsf{msg} \xleftarrow{\$} \{0,1\};\\ \mathsf{ct} \leftarrow \mathsf{Enc}(\lambda,\mathsf{stmt},\mathsf{msg})\end{array}\right] \geq 1/2 + 1/q(|\mathsf{stmt}|)$$
$$\implies \Pr[\mathsf{Ext}(\mathsf{stmt},\mathsf{aux}) = \mathsf{wit} : (\mathsf{stmt},\mathsf{wit}) \in \mathcal{R}] \geq 1/p(|\mathsf{stmt}|)$$

Theorem 5 (Extractable Security). *Let* $\mathcal{L}$ *be a language recognized by a read-once formula* $\mathcal{C}$ *over the alphabet* Λ*. Given that* $\mathsf{SKE} = (\mathsf{Enc},\mathsf{Dec})$ *is modeled as an ideal cipher with parameters* $\ell_{\underline{\mathsf{ct}}}$ *and* $\ell_{\underline{\mathsf{sk}}}$*, the witness encryption scheme* $\mathsf{WE} = (\mathsf{Enc},\mathsf{Dec})$ *for* $\mathcal{C}$ *is extractable secure against every* **PPT** *adversary* A *with extraction error at most* $\frac{\mathsf{q}^2}{2^{\ell_{\underline{\mathsf{ct}}}}} + \frac{\mathsf{q}^2}{2^{\ell_{\underline{\mathsf{sk}}}}}$*, where* q *is the number of all queries to* SKE*.*

Proof. Let A be a **PPT** adversary against the extractable security property of the $\mathsf{WE} = (\mathsf{Enc},\mathsf{Dec})$ scheme. We will start the proof by describing an extractor Ext, which is given as input a statement stmt and exploits A to compute the witness that $\mathsf{stmt} \in \mathcal{L}$. First the extractor represents stmt as a read-once formula $\mathcal{C}$ over alphabet Λ such that $\mathcal{C}(\mathsf{stmt},.) = 1$ iff there exists a witness wit testifying $\mathsf{stmt} \in \mathcal{L}$. Without loss of generality, assume all gates of the formula $\mathcal{C}$ have fan-in τ.

The extractor executes the $\mathsf{WE.Enc}(\mathsf{stmt},\mathsf{msg})$ algorithm according to specification. Ext registers the association between the labels it chooses and the values of the wires for each gate g it encodes. We will denote this register in the $\mathcal{Q}_{\mathsf{Chal}}$ table. Ext sets up also a challenge table to register A's queries. Let $\mathsf{ct}_{\mathsf{chal}}$ be the witness encryption ciphertext computed by Ext. The extractor Ext runs the adversary A on input stmt and $\mathsf{ct}_{\mathsf{chal}}$.

A might query the SKE oracle. If due to any of A's queries, there is a collision in SKE, then Ext aborts. This event may happen with probability at most $\frac{\mathsf{q}^2}{2^{\ell_{\underline{\mathsf{ct}}}}}$. If A makes a direct query to SKE or AFHE on an input x such that x contains a substring $\underline{K} \in \{0,1\}^{\ell_{\underline{\mathsf{sk}}}}$ and $\underline{K}$ can be found in $\mathcal{Q}_{\mathsf{Chal}}$ but isn't in $\mathcal{Q}_{\mathsf{A}}$ yet, then Ext aborts. In other words, if A guesses a key. Since A does not get any information on the keys stored in $\mathcal{Q}_{\mathsf{Chal}}$, this event may happen with probability at most $\mathsf{q}^2/2^{\ell_{\underline{\mathsf{sk}}}}$. Note that we upperbound $\rho \cdot \mathsf{q} \leq \mathsf{q}^2$, where $\rho \in \mathbb{N}$ denotes the number of all SKE keys used in WE. When A queries the oracle $\mathsf{SKE.Dec}$ on input x, Ext does the following.

- If x is not of the form $(\underline{K}_1, \ldots, \underline{K}_\tau, \underline{\mathsf{ct}}) \in \{0,1\}^{\tau\cdot\ell_{\underline{\mathsf{sk}}}+\ell_{\underline{\mathsf{ct}}}}$, or $x = (\underline{K}_1, \ldots, \underline{K}_\tau, \underline{\mathsf{ct}}) \in \{0,1\}^{\tau\cdot\ell_{\underline{\mathsf{sk}}}+\ell_{\underline{\mathsf{ct}}}}$ but x is not in the $\mathcal{Q}_{\mathsf{Chal}}$ table then Ext ignores the query.

– Otherwise, Ext looks up $\mathcal{Q}_{\mathsf{Chal}}$ for the corresponding tuple $\underline{K}_1, \ldots, \underline{K}_\tau, \underline{\mathsf{ct}}, \mathsf{msg}$) and stores it in the $\mathcal{Q}_{\mathsf{A}}$ table. Furthermore, for each $j \in [\tau]$, Ext checks whether r_j was returned by a previous query to $\mathsf{SKE.Dec}$. Concretely, Ext checks whether there is a tuple in the $\mathcal{Q}_{\mathsf{A}}$ table containing r_j.
  - If $\underline{K}_j$ was in the $\mathcal{Q}_{\mathsf{A}}$ table, then
    - If $\underline{K}_j$ is a leave, Ext stops the recursion and proceeds further.
    - If $\underline{K}_j$ is not a leave, Ext runs the same procedure recursively for $\underline{K}_j$.
  - If $\underline{K}_j$ was not in the $\mathcal{Q}_{\mathsf{A}}$ table, then Ext fails at extracting the witness and aborts. Since the only way A may learn any information about $\underline{K}_j$ is either obtaining it from Ext or from a previous SKE oracle query, $\underline{K}_j$ must have been guessed by A. However, we already ruled out the possibility.

Assume now that Ext did not abort the experiment and A returns the message msg. Ext will now check whether msg was returned by the $\mathsf{SKE.Dec}$ oracle, as described above, and simultaneously reconstruct the witness wit. Since Ext did not abort previously, we have that each label was either returned by $\mathsf{SKE.Dec}$ or is a leave. The extractor will run the following procedure for each gate recursively. Suppose Ext starts with an output wire K_{out}. Ext looks up the $\mathcal{Q}_{\mathsf{A}}$ table for tuples $(.,\ldots,.,\underline{\mathsf{ct}},\underline{K}_{\mathsf{out}})$. If there are multiple tuples of this form, Ext chooses one of them at random. Denote the chosen tuple as $(\underline{K}_1^{v_1},\ldots,\underline{K}_\tau^{v_\tau},\underline{\mathsf{ct}},\underline{K}_{\mathsf{out}})$ which corresponds to gate g. Now, Ext finds the wire values $(v_1,\ldots,v_\tau)$ and v_{out} corresponding to the label and sets the part of the witness that is the values of g's wires to the extracted wire values. Note, that from correctness of $\mathsf{WE.Dec}$ we have $\mathsf{g}(v_1,\ldots,v_\tau) = v_{\mathsf{out}}$, where g is the function corresponding to the input and output wires. Next, Ext repeats this procedure for every $\underline{K}_1^{v_1}, \ldots, \underline{K}_\tau^{v_\tau}$ except any of the labels correspond to input wires. Finally, Ext will start extraction from the output wire msg, which corresponds to the value accept and ends the execution upon reaching the input wires.

To sum up, the extractor Ext extracts a valid witness wit with probability at least $1 - (\frac{\mathsf{q}^2}{2^{\ell_{\underline{\mathsf{ct}}}}} + \frac{\mathsf{q}^2}{2^{\ell_{\underline{\mathsf{sk}}}}})$.

Remark 5. Note that the security of witness encryption as given by Definition 13, reduces the message to only a single bit. However, from the proof of Theorem 5, it is immediate that the definition can be extended to messages from the entire domain of $\ell_{\underline{\mathsf{ct}}}$.

B Full Correctness Analysis of the Annihilating (Leveled) Fully Homomorphic Encryption Candidate

In this section we give the analysis of the homomorphic operations that constitute Theorem 1. We do so by showing that each operation is correct. First we recall the definition of B-Bounded distributions, and introduce our notation for phase and error. Additionally, we will denote $\gamma_{\mathcal{R}}$ the expansion factor of polynomials in the ring $\mathcal{R} = \mathbb{Z}[X]/(X^N + 1)$. Recall that by the Cauchy-Schwarz theorem the expansion factor $\gamma_{\mathcal{R}}$ of $\mathcal{R}$ is bounded by $\sqrt{N}$.

Definition 14 (B-Bounded Distributions). *A distribution ensemble $\{\mathcal{X}_B\}_{B\in\mathbb{N}}$ is called B-Bounded if*

$$\Pr_{e\leftarrow\mathcal{X}_B}[||e||_2 \geq B] \leq \mathsf{negl}(\lambda).$$

Definition 15 (Phase and Error for GLWE samples). *We define the phase of a sample $\mathbf{c} = \mathsf{GLWE}_{\mathcal{X},n,N,q}([\mathbf{s}_i]_{i=1}^d, \mathfrak{m})$, as $\mathsf{Phase}(\mathbf{c}) = [1, [\mathbf{s}]_{i=1}^d]\cdot\mathbf{c}^\top$. We define the error of a GLWE sample as $\mathsf{Error}(\mathbf{c}) = \mathsf{Phase}(\mathbf{c}) - \mathfrak{m}$. Furthermore, we note that if $\mathcal{X}$ is B-bounded, then $||\mathsf{Error}(\mathbf{c})||_2 \leq B$.*

Lemma 1 (Linear Homomorphism of GLWE samples). *Let us define $\mathbf{c} = \mathsf{GLWE}_{\mathcal{X}_{B_\mathbf{c}},n,N,q}([\mathbf{s}_i]_{i=1}^d, \mathfrak{m}_\mathbf{c})$ and $\mathbf{d} = \mathsf{GLWE}_{\mathcal{X}_{B_\mathbf{d}},n,N,q}([\mathbf{s}_i]_{i=1}^d, \mathfrak{m}_\mathbf{d})$, where $\mathcal{X}_{B_\mathbf{c}}$ and $\mathcal{X}_{B_\mathbf{d}}$ is respectively a $B_\mathbf{c}$ and $B_\mathbf{d}$-Bounded distribution. If $\mathbf{c}_\mathsf{out} \leftarrow \mathbf{c} + \mathbf{d}$, then $\mathbf{c}_\mathsf{out} \in \mathsf{GLWE}_{\mathcal{X}_B,n,N,q}([\mathbf{s}]_{i=1}^d, \mathfrak{m})$, where $\mathfrak{m} = \mathfrak{m}_\mathbf{c} + \mathfrak{m}_\mathbf{d}$, and $\mathcal{X}_B$ is a B-Bounded distribution with $B \leq B_\mathbf{c} + B_\mathbf{d}$. Furthermore, let $\mathfrak{d} \in \mathcal{R}_{B_\mathfrak{d}}$ where $B_\mathfrak{d} \in \mathbb{N}$. If $\mathbf{c}_\mathsf{out} \leftarrow \mathbf{c}\cdot\mathfrak{d}$, then $\mathbf{c}_\mathsf{out} \in \mathsf{GLWE}_{\mathcal{X}_B,n,N,q}([\mathbf{s}]_{i=1}^d, \mathfrak{m}_\mathbf{c}\cdot\mathfrak{d})$, and $\mathcal{X}_B$ is a B-Bounded distribution with $B \leq N\cdot B_\mathbf{c}\cdot(B_\mathfrak{d}-1)$.*

Proof. Denote $\mathbf{c} = [\mathfrak{b}_\mathbf{c}, \mathbf{a}_\mathbf{c}] \in \mathcal{R}_q^{1+n\cdot d}$, where $\mathfrak{b}_\mathbf{c} = \langle\mathbf{a}_\mathbf{c}, [\mathbf{s}_i]_{i=1}\rangle + \mathfrak{m}_\mathbf{c} + \mathfrak{e}_\mathbf{c}$ and $\mathbf{d} = [\mathfrak{b}_\mathbf{d}, \mathbf{a}_\mathbf{d}] \in \mathcal{R}_q^{1+n\cdot d}$, where $\mathfrak{b}_\mathbf{d} = \langle\mathbf{a}_\mathbf{d}, [\mathbf{s}_i]_{i=1}\rangle + \mathfrak{m}_\mathbf{d} + \mathfrak{e}_\mathbf{d}$. Then we have $\mathbf{c}_\mathsf{out} = [\mathfrak{b}, \mathbf{a}] = \mathbf{c}+\mathbf{d} = [\mathfrak{b}_\mathbf{c}+\mathfrak{b}_\mathbf{d}, (\mathbf{a}_\mathbf{c}+\mathbf{a}_\mathbf{d})]$, and $\mathfrak{b} = \langle\mathbf{a}, \mathbf{s}_i]_{i=1}\rangle + \mathfrak{m}+\mathfrak{e}$, where $\mathfrak{m} = \mathfrak{m}_\mathbf{c}+\mathfrak{m}_\mathbf{d}$ and $\mathfrak{e} = \mathfrak{e}_\mathbf{c} + \mathfrak{e}_\mathbf{d}$. Thus, we have that $\mathsf{Error}(\mathbf{c}_\mathsf{out}) = \mathfrak{e}_\mathbf{c} + \mathfrak{e}_\mathbf{d}$, and $||\mathsf{Error}(\mathbf{c}_\mathsf{out})||_2 \leq B \leq ||\mathfrak{e}_\mathbf{c}+\mathfrak{e}_\mathbf{d}||_2 = ||\mathfrak{e}_\mathbf{c}||_2 + ||\mathfrak{e}_\mathbf{d}||_2 \leq B_\mathbf{c}+B_\mathbf{d}$. For $\mathfrak{d} \in \mathcal{R}_{B_\mathfrak{d}}$ and $\mathbf{c}_\mathsf{out} = \mathbf{c}\cdot\mathfrak{d} = [\mathfrak{b}_\mathfrak{d}, \mathbf{a}_\mathfrak{d}]$, where $\mathbf{a}_\mathfrak{d} = \mathbf{a}_\mathbf{d}\cdot\mathfrak{d}$ and $\mathfrak{b}_\mathfrak{d} = \langle\mathbf{a}_\mathfrak{d}, [\mathbf{s}_i]_{i=1}^d\rangle + \mathfrak{m}_\mathbf{c}\cdot\mathfrak{d} + \mathfrak{e}_\mathbf{c}\cdot\mathfrak{d}$. Thus $||\mathsf{Error}(\mathbf{c}_\mathsf{out})||_2 \leq B \leq ||\mathfrak{e}_\mathbf{c}\cdot\mathfrak{d}||_2 = \gamma_\mathcal{R}\cdot||\mathfrak{e}||_2\cdot||\mathfrak{d}||_2 \leq \sqrt{N}\cdot B_\mathbf{c}\cdot(B_\mathfrak{d}-1)\cdot\sqrt{N} \leq N\cdot B_\mathbf{c}\cdot(B_\mathfrak{d}-1)$.

Lemma 2 (Correctness of Multiplication). *Let $\hat{\mathsf{ct}} = (\hat{\mathbf{c}}, \mathcal{T})$ and $\hat{\mathsf{ct}}' = (\hat{\mathbf{c}}', \mathcal{T})$ where $\mathbf{c} = \mathsf{GLWE}_{\mathcal{X}_B,n,N,q}(\mathbf{s}, \mathfrak{m})$ and $\mathbf{c}' = \mathsf{GLWE}_{\mathcal{X}_{B'},n,N,q}(\mathbf{s}, \mathfrak{m}')$, where $\mathcal{X}_B$ and $\mathcal{X}_{B'}$ are B and B' bounded distributions respectively. Let $B_{\mathsf{G}^{-1}}$ be the bound on the output elements of the decomposition function $\mathsf{G}_{B,q}^{-1}$ and denote $\ell = \lceil\log_B q\rceil$. Let $\mathcal{X}_{\overline{\mathsf{ek}}}$ be the $B_{\overline{\mathsf{ek}}}$ distribution of the GLWE sample in the relinearization key $\overline{\mathsf{ek}}$. Let $\overline{\mathsf{ct}}_\mathsf{out}$ be the outcome of multiplying $\hat{\mathsf{ct}}$ and $\hat{\mathsf{ct}}'$, where $\overline{\mathsf{ct}} = (\mathbf{c}_\mathsf{out}, \mathcal{T})$. Then $\mathbf{c}_\mathsf{out} \in \mathsf{GLWE}_{\mathcal{X}_{B_\mathsf{out}},n,N,q}(\mathbf{s}, \mathfrak{m}_\mathsf{out})$, where $\mathfrak{m}_\mathsf{out} = \mathfrak{m}_\mathbf{c}\cdot\mathfrak{m}_\mathbf{d}$ and the bound is*

$$B_\mathsf{out} \leq B_\otimes + \gamma_\mathcal{R}\cdot B_{\mathsf{G}^{-1}}\cdot B_{\overline{\mathsf{ek}}}\cdot(\ell\cdot n\cdot|\mathcal{T}|\cdot(1+n+n\cdot|\mathcal{T}|))$$

where $B_\otimes \leq \gamma_\mathcal{R}\cdot(||\mathfrak{m}||_2\cdot B' + ||\mathfrak{m}'||_2\cdot B + B\cdot B')$.

Proof. Let us first recall how computing the Kronecker product of two ciphertexts works. Let $\hat{\mathbf{c}} = [\hat{\mathfrak{b}}, [\hat{\mathbf{a}}_k]_{k\in\mathcal{T}}]$ and $\hat{\mathbf{c}}' = [\hat{\mathfrak{b}}', [\hat{\mathbf{a}}'_k]_{k\in\mathcal{T}}]$. Let $\tilde{\mathbf{c}} = \hat{\mathbf{c}}\otimes\hat{\mathbf{c}}'$, be the extended ciphertext. We have that $\hat{\mathbf{c}}$ is a correct ciphertext with respect to $\tilde{\mathbf{s}} = \hat{\mathbf{s}}\otimes\hat{\mathbf{s}}$, where $\hat{\mathbf{s}} = [1, [\mathbf{s}_k]_{k\in\mathcal{T}}]$. To see this, note that from the mixed-product property of the Kronecker product we have $\langle\tilde{\mathbf{c}}, \tilde{\mathbf{s}}\rangle = \langle\hat{\mathbf{c}}\otimes\hat{\mathbf{c}}', \hat{\mathbf{s}}\otimes\hat{\mathbf{s}}\rangle = \langle\hat{\mathbf{c}}, \hat{\mathbf{s}}\rangle\cdot\langle\hat{\mathbf{c}}', \hat{\mathbf{s}}\rangle = (\mathfrak{m}+\mathfrak{e})\cdot(\mathfrak{m}'+\mathfrak{e}')$. Hence, we can see that

$$\begin{aligned}\mathsf{Error}(\hat{\mathbf{c}}\otimes\hat{\mathbf{c}}') &= ||\mathfrak{m}\cdot\mathfrak{e}' + \mathfrak{m}'\cdot\mathfrak{e} + \mathfrak{e}\cdot\mathfrak{e}'||_2 \leq B_\otimes \\ &\leq \gamma_\mathcal{R}\cdot(||\mathfrak{m}||_2\cdot B' + ||\mathfrak{m}'||_2\cdot B + B\cdot B').\end{aligned}$$

Now consider the relinearization step. Note that we can rewrite line (1).

$$\begin{aligned}
&\sum_{k\in\mathcal{T}}\sum_{i=1}^{n}\mathsf{G}^{-1}_{B,q}(\hat{\mathfrak{b}}'\cdot\hat{\mathbf{a}}_k[i]+\hat{\mathbf{a}}'_k[i]\cdot\hat{\mathfrak{b}})\cdot\hat{\mathsf{ek}}_{1,k}[i,*]^\top\\
=\;&\mathsf{GLWE}_{\mathcal{X}_1,n,N,q}(\hat{\mathfrak{b}}\cdot\hat{\mathfrak{b}}'+\sum_{k\in\mathcal{T}}\sum_{i=1}^{n}\hat{\mathfrak{b}}'\cdot\hat{\mathbf{a}}_k[i]+\hat{\mathbf{a}}'_k[i]\cdot\hat{\mathfrak{b}}\cdot\mathfrak{s}_k)\\
=\;&\mathsf{GLWE}_{\mathcal{X}_1,n,N,q}(\hat{\mathfrak{b}}\cdot\hat{\mathfrak{b}}'+\hat{\mathfrak{b}}'\cdot\langle[\hat{\mathbf{a}}_k]_{k\in\mathcal{T}},[\mathbf{s}_k]_{k\in\mathcal{T}}\rangle)+\hat{\mathfrak{b}}\cdot\langle[\hat{\mathbf{a}}'_k]_{k\in\mathcal{T}},[\mathbf{s}_k]_{k\in\mathcal{T}}\rangle) \qquad (7)
\end{aligned}$$

Then we have that the sum of the lines 2 and 3 of the relinearization step is equal to a $\mathsf{GLWE}_{\mathcal{X}_{2,3},n,N,q}$ sample with some error distribution $\mathcal{X}_{2,3}$ of the message:

$$\begin{aligned}
&\sum_{k\in\mathcal{T}}\sum_{\substack{i=1,\\ i'=1}}^{n}\hat{\mathbf{a}}_k[i]\cdot\hat{\mathbf{a}}'_k[i']\cdot\mathbf{s}_k[i]\cdot\mathbf{s}_k[i']+\sum_{\substack{k,j\in\mathcal{T},\\ k\neq j}}\sum_{\substack{i=1,\\ i'=1}}^{n}\hat{\mathbf{a}}_k[i]\cdot\hat{\mathbf{a}}'_j[i']\cdot\mathbf{s}_k[i]\cdot\mathbf{s}_k[i']\\
=&\sum_{k,j\in\mathcal{T}}\sum_{\substack{i=1,\\ i'=1}}^{n}\hat{\mathbf{a}}_k[i]\cdot\hat{\mathbf{a}}'_j[i']\cdot\mathbf{s}_k[i]\cdot\mathbf{s}_k[i']\\
=&\langle[\hat{\mathbf{a}}_k]_{k\in\mathcal{T}}\otimes[\hat{\mathbf{a}}_k]_{k\in\mathcal{T}},[\hat{\mathbf{s}}_k]_{k\in\mathcal{T}}\otimes[\hat{\mathbf{s}}_k]_{k\in\mathcal{T}}\rangle \qquad (8)
\end{aligned}$$

Finally, from the sum of 7 and 8 we have

$$\begin{aligned}
&\mathsf{GLWE}_{\mathcal{X}_{\mathsf{out}},n,N,q}(\hat{\mathfrak{b}}\cdot\hat{\mathfrak{b}}'+\hat{\mathfrak{b}}'\cdot\langle[\hat{\mathbf{a}}_k]_{k\in\mathcal{T}},[\mathbf{s}_k]_{k\in\mathcal{T}}\rangle)+\hat{\mathfrak{b}}\cdot\langle[\hat{\mathbf{a}}'_k]_{k\in\mathcal{T}},[\mathbf{s}_k]_{k\in\mathcal{T}}\rangle\\
&\qquad+\langle[\hat{\mathbf{a}}_k]_{k\in\mathcal{T}}\otimes[\hat{\mathbf{a}}_k]_{k\in\mathcal{T}},[\hat{\mathbf{s}}_k]_{k\in\mathcal{T}}\otimes[\hat{\mathbf{s}}_k]_{k\in\mathcal{T}}\rangle)\\
=\;&\mathsf{GLWE}_{\mathcal{X}_{\mathsf{out}},n,N,q}(\langle[\hat{\mathfrak{b}},[\hat{\mathbf{a}}_k]_{k\in\mathcal{T}}]\otimes[\hat{\mathfrak{b}}',[\hat{\mathbf{a}}_k]_{k\in\mathcal{T}}],[1,[\hat{\mathbf{s}}_k]_{k\in\mathcal{T}}]\otimes[1,[\hat{\mathbf{s}}_k]_{k\in\mathcal{T}}]\rangle)\\
=\;&\mathsf{GLWE}_{\mathcal{X}_{\mathsf{out}},n,N,q}(\langle\hat{\mathsf{ct}}\otimes\hat{\mathsf{ct}}',\hat{\mathsf{sk}}\otimes\hat{\mathsf{sk}}\rangle)\\
=\;&\mathsf{GLWE}_{\mathcal{X}_{\mathsf{out}},n,N,q}(\langle\hat{\mathsf{ct}},\hat{\mathsf{sk}}\rangle\cdot\langle\hat{\mathsf{ct}}',\hat{\mathsf{sk}}\rangle)\\
=\;&\mathsf{GLWE}_{\mathcal{X}_{\mathsf{out}},n,N,q}(\mathfrak{m}_1\cdot\mathfrak{m}_2)
\end{aligned}$$

Note that the relinearization consists of linear operations on GLWE ciphertexts, so the error bound follows from counting the operations. Finally, we have that $\mathcal{X}_{\mathsf{out}}$ is a B_{out} bounded distribution where

$$B_{\mathsf{out}}\leq B_{\otimes}+\gamma_{\mathcal{R}}\cdot B_{\mathsf{G}^{-1}}\cdot B_{\overline{\mathsf{ek}}}\cdot(\ell\cdot n\cdot|\mathcal{T}|\cdot(1+n+n\cdot|\mathcal{T}|)).$$

C Annihilating FHE: The Full Scheme

Let us first note that when tags are non-annihilating, particularly for $\mathcal{K}=\emptyset$, our scheme is equivalent to the basic version of the BGV scheme. In particular, when all tags are non-annihilating, then we can rewrite the GLWE samples as $\mathsf{GLWE}_{\mathcal{X},\kappa\cdot n,N,q}(\mathbf{s},.)$ where $\mathbf{s}=[\mathbf{s}_k]_{k=1}^{\kappa}\in\mathcal{R}_q^{\kappa\cdot n}$ instead of $\mathsf{GLWE}_{\mathcal{X},n,N,q}([\mathbf{s}_k]_{k=1}^{\kappa},.)$ as in the protocol. In other words a multikey GLWE sample is equivalent to a "classic" GLWE sample, with a higher dimension $n'=\kappa\cdot n$. The change is only

notational. Furthermore, we may think of the multikey relinearisation keys mk as a part of the BGV relinearisation key $\mathsf{ek}_{2,k}$ in the single key mode. Thus, the only real difference between BGV/BFV-type cryptosystems and our multikey version is that we do not publish certain parts of the relinearization key and use only a fraction of the secret key for encryption. Otherwise, the analysis and numerous optimizations for BGV/BFV-like cryptosystems also apply to our scheme. We refer to the work of Costache and Smart [CS16] for an excellent comparison of different variants of BGV/BFV-type cryptosystems.

The full scheme is as follows. The Setup algorithm takes as input additionally the level L. For each level $h \in [L]$ we generate a moduli q_h where $q_h \geq g_{h-1}$, and independent secret keys $\mathbf{s}_k^{(h)} \xleftarrow{\$} \mathcal{R}_{q_h}^{n\cdot\kappa}$. As for the relinearization keys, we encrypt the keys of level h under keys of level $h-1$. Specifically, we modify steps 3 to 5.1 of the Setup is as follows:

3. For all $k \in [\kappa]$ and $h \in [L-1]$ do:
   3.1. Choose the secret keys as described above (for all levels $h \in [L]$).
   3.2. For all $i \in [n]$ and $l \in [\ell]$ compute

$$\mathsf{ek}_{1,k}[i,l,h] \xleftarrow{\$} \mathsf{GLWE}_{\mathcal{X}_{\mathsf{ek}},n,N,q_h}(\mathbf{s}_k^{(h)}, \mathbf{g}[l]\cdot\mathbf{s}_k^{(h+1)}[i]).$$

   3.3. For all $i, i' \in [n]$ and $l \in [\ell]$ compute

$$\mathsf{ek}_{2,k}[i,i',l,h] \xleftarrow{\$} \mathsf{GLWE}_{\mathcal{X}_{\mathsf{ek}},n,N,q_h}(\mathbf{s}_k^{(h)}, \mathbf{g}[l]\cdot\mathbf{s}_k^{(h+1)}[i]\cdot\mathbf{s}_k^{(h+1)}[i']).$$

4. Define the non-annihilating tags $\overline{\mathcal{K}} = ([\kappa]\times[\kappa]) \setminus \mathcal{K}$.
5. For all $(k,j) \in \overline{\mathcal{K}}$ do:
   5.1. For all $i, i' \in [n]$ and $l \in [\ell]$ set

$$\mathsf{mk}_{k,j}[i,i',l] \xleftarrow{\$} \mathsf{GLWE}_{\mathcal{X}_{\mathsf{ek}},n,N,q_h}([\mathbf{s}_k^{(h)}, \mathbf{s}_j^{(h)}], \mathbf{g}[l]\cdot\mathbf{s}_k^{(h+1)}[i]\cdot\mathbf{s}_j^{(h+1)}[i']).$$

To encrypt a message, we use the secret key at level L. The addition of two ciphertexts at the same level is done as in the basic scheme. When multiplying ciphertexts at a level h, we obtain a ciphertext at level $h-1$ after relinearization. In its simplest version, the moduli at each level are equivalent. However, to achieve better noise control, Brakerski, Vaikuntanathan [BV11] showed a technique called modulus switching. In the modulus switching technique, we choose the sequence of moduli very carefully and change the moduli of ciphertext by multiplying it with $\frac{q_{h-1}}{q_h}$ and rounding at each level. This way, we can evaluate arbitrary polynomial-size circuits without resorting to bootstrapping and circular security. Finally, we note that to realize the cycle tester as given by Construction 2, we set the circuit to be obfuscated as the decryption function at the last level of the AFHE.

Remark 6 (Redundant Relinearization Keys). Note that to simplify exposition, we include multikey relinearization keys that are not needed. Specifically, the keys $\mathsf{mk}_{k,j}$ and $\mathsf{mk}_{j,k}$ for $(k,j) \in \overline{\mathcal{K}}$ encrypt for the same plaintexts, and it is enough to publish only one of them.

As mentioned in Section 3.2, both the least significant LSB and most significant bit MSB encodings of the message are possible. In LSB $\mathfrak{m} \in \mathcal{R}_p$, for some $p \leq q$, and $\mathfrak{e} = p \cdot \mathfrak{e}'$. Then decryption outputs $\langle \mathbf{c}, [1, [\mathbf{s}_k]_{k=1}^{\kappa}\rangle] \mod p$. In MSB we scale the message $\mathfrak{m} \cdot \frac{q}{p}$, and have $||\mathfrak{e}||_2 \leq \frac{q}{2 \cdot p}$. In this case decryption is computed as $\lfloor \frac{p}{q} \cdot \langle \mathbf{c}, [1, [\mathbf{s}_k]_{k=1}^{\kappa}]\rangle \rceil$, where $\lfloor . \rceil$ outputs the closes integer. The MSB encoding is often referred to as the scale-invariant version [Bra12, FV12], but we note that multiplication requires to rescale and round the ciphertexts. In particular step 8 of the multiplication in Construction 1 is

$$\begin{aligned}\mathbf{c}_{\text{out}} = [\lfloor \frac{p}{q} \cdot (\hat{\mathfrak{b}} \cdot \hat{\mathfrak{b}}') \rceil, \mathbf{0}] &+ \sum_{k \in \mathcal{T}} \sum_{i=1}^{n} \mathsf{G}_{B,q}^{-1}\Big(\lfloor \frac{p}{q} \cdot (\hat{\mathfrak{b}}' \cdot \hat{\mathbf{a}}_k[i] + \hat{\mathbf{a}}'_k[i] \cdot \hat{\mathfrak{b}}) \rceil \Big) \cdot \hat{\mathsf{ek}}_{1,k}[i, *]^{\top} \\ &+ \sum_{k \in \mathcal{T}} \sum_{\substack{i=1, \\ i'=1}}^{n} \mathsf{G}_{B,q}^{-1}\Big(\lfloor \frac{p}{q} \cdot (\hat{\mathbf{a}}_k[i] \cdot \hat{\mathbf{a}}'_k[i']) \rceil \Big) \cdot \hat{\mathsf{ek}}_{2,k}[i, i', *]^{\top} \\ &+ \sum_{\substack{k,j \in \mathcal{T}, \\ k \neq j}} \sum_{\substack{i=1, \\ i'=1}}^{n} \mathsf{G}_{B,q}^{-1}\Big(\lfloor \frac{p}{q} \cdot (\hat{\mathbf{a}}_k[i] \cdot \hat{\mathbf{a}}'_j[i']) \rceil \Big) \cdot \hat{\mathsf{mk}}_{k,j}[i, i', *]^{\top},\end{aligned}$$

where $\frac{p}{q} \cdot (\hat{\mathfrak{b}} \cdot \hat{\mathfrak{b}}')$, $\frac{p}{q} \cdot (\hat{\mathfrak{b}}' \cdot \hat{\mathbf{a}}_k[i] + \hat{\mathbf{a}}'_k[i] \cdot \hat{\mathfrak{b}})$ and $\frac{p}{q} \cdot (\hat{\mathbf{a}}_k[i] \cdot \hat{\mathbf{a}}'_j[i'])$ are performed in $\mathcal{R}$ instead of $\mathcal{R}_q$. In other words, we reduce the coefficients modulo q after rescaling and rounding.

C.1 Semantic Security of the Annihilating FHE Candidate

Let us first recall the definition for semantic security.

Definition 16 (Semantic Security). *We say that* $\mathsf{AFHE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ *is semantically-secure, if for all security parameters* $\lambda \in \mathbb{N}$, *all* $\kappa = \mathsf{poly}(\lambda)$, *all* $\mathcal{K} \subseteq [\kappa] \times [\kappa]$ *and all* **PPT** *adversaries* $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2)$, *we have that*

$$\left| \Pr \left[b = \mathsf{A}_2(\overline{\mathsf{ct}}, \mathsf{st}) : \begin{array}{c} (\overline{\mathsf{ek}}, \overline{\mathsf{sk}}) \leftarrow \mathsf{Setup}(\lambda, \kappa, \mathcal{K}) \\ (\mathsf{msg}_0, \mathsf{msg}_1, \mathsf{st}) \leftarrow \mathsf{A}_1(\lambda, \overline{\mathsf{ek}}); \\ b \xleftarrow{\$} \{0,1\}; \\ \overline{\mathsf{ct}} \leftarrow \mathsf{Enc}(\overline{\mathsf{sk}}, \mathcal{T}, \mathsf{msg}_b); \end{array} \right] - \frac{1}{2} \right| \leq \mathsf{negl}(\lambda),$$

where the probability is over the random choice of b *and random coins of* Setup *and* Enc.

The security proof is a standard hybrid argument in which we first eliminate all evaluation keys, starting from level 1, and then change the challenge encryption assuming GLWE. Finally, we note that to show the basic scheme's security, we need to assume circular GLWE since we have a 1-cycle. By 1-cycle, we mean that a function on the keys $\mathbf{s}_k$ is encrypted under $\mathbf{s}_k$ in the relinearisation keys.

Theorem 6 (Security). *The full scheme is semantically secure, given security of* GLWE.

Proof (sketch). The proof is via the following hybrid argument. We note that we need to iterate the hybrids starting from $h = 1$ to L. However, for other iterators, we do not require any particular order.

Hybrid 0: This is the original scheme.

Hybrid $(1, h, k, i, l)$**:** For all $h \in [L]$, starting from $h = 1$, we consider the hybrids **Hybrid** $(1, h, k, i, l)$ where $\mathsf{ek}_{1,k}[i, l, h]$ is computed at random and $k \in [\kappa]$, $i \in [n]$ and $l \in [\ell]$. The indistinguishability of the hybrids follows trivially from the GLWE assumption.

Hybrid $(2, h, k, i, i', l)$**:** For all $h \in [L]$, starting from $h = 1$, we consider the hybrids **Hybrid** $(2, h, k, i, i', l)$ where $\mathsf{ek}_{2,k}[i, i', l, h]$ is computed at random and $k \in [\kappa]$, $i, i' \in [n]$ and $l \in [\ell]$. The indistinguishability of the hybrids follows trivially from the GLWE assumption.

Hybrid $(3, h, i, i', l)$**:** For all $h \in [L]$, starting from $h = 1$, we consider the hybrids **Hybrid** $(3, h, i, i', l)$ where $\mathsf{mk}_{k,j}[i, i', l]$ is computed at random and $(k, j) \in \overline{\mathcal{K}}$, $i, i' \in [n]$ and $l \in [\ell]$. The indistinguishability of the hybrids follows trivially from the GLWE assumption.

Hybrid 4: In **Hybrid** 4, we compute the challenge ciphertext at random. The indistinguishability follows from the GLWE assumption.

C.2 Semantic Security in the Presence of a Cycle Tester

Below we give the semantic security proof for the circular insecure annihilating fully homomorphic encryption given by Construction 2. The proof is basically a rewrite of the semantic security proofs for the cycle testers given in [GKW17a].

Theorem 7 (Semantic Security of Construction 2). *Let* AFHE *be a semantically security annihilating encryption scheme, and* $\mathsf{lockObf} = (\mathsf{Obf}, \mathsf{Eval})$ *be a lockable obfuscation. Let* ctAFHE *be an annihilating fully homomorphic encryption with a cycle tester build from* AFHE *and* lockObf *as specified by Construction 2. Given semantic security of* AFHE *and distributional virtual black-box security of* $\mathsf{lockObf} = (\mathsf{Obf}, \mathsf{Eval})$, ctAFHE *is semantically secure.*

Proof (Sketch). We show the proof via a sequence of hybrid experiments.

Hybrid 0: This is the original scheme in which we encrypt the message m_0.

Hybrid 1: This hybrid is as the previous, except we simulate the lockable obfuscation. Specifically, we compute $\tilde{C} \leftarrow \mathsf{Sim}(\lambda, \eta, \nu, \xi)$ instead of $\tilde{C} \leftarrow \mathsf{lockObf.Obf}(\lambda, P, \mathsf{lock})$. Indistinguishability between **Hybrid 0** and **Hybrid 1** follows from distributional virtual black-box security of the lockable obfuscation scheme. Note that we may reduce security to DVBB since lock is chosen independently from the uniform distribution.

Hybrid 2: This hybrid is as the previous, except we encrypt the message m_1. Since the lockable obfuscation is simulated, indistinguishability between **Hybrid 1** and **Hybrid 2** follows from semantic security of the annihilating fully homomorphic scheme.

C.3 Instantiating the Lockable Obfuscation

We recall the construction of the GGH15-based lockable obfuscation [CVW18] and argue that when we instantiate the cycle tester with this concrete lockable obfuscator, then ideal cipher annihilation security should hold. We note that we limit the exposition to the algorithms and preliminaries necessary to understand our arguments. For detailed correctness and security proofs, we refer to [CVW18].

Insecure Instantiation We first show an example of an insecure instantiation[4]. We use the example to showcase the nature of the problem in instantiating the lockable obfuscation and as a tool to further argue the security of our candidate.

Theorem 8. *Let* $\mathsf{AFHE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ *be an annihilating homomorphic encryption scheme, and let* $\mathsf{FHE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ *be a fully homomorphic encryption scheme. Let* $\mathsf{lockObf}$ *be a lockable obfuscation scheme. We construct an alternative lockable obfuscation* $\mathsf{lockObf}' = (\mathsf{Obf}, \mathsf{Eval})$ *for the function* $\mathsf{AFHE.Dec}(\overline{\mathsf{sk}}, .)$*, where* $\mathsf{lockObf}'.\mathsf{Obf}(C, \mathsf{lock})$ *returns* $\tilde{C}' = (\tilde{C}, \mathsf{ct}_{\mathsf{FHE},\overline{\mathsf{sk}}})$ *with* $\tilde{C} \leftarrow \mathsf{lockObf.Obf}(C, \mathsf{lock})$ *and* $\mathsf{ct}_{\mathsf{FHE},\overline{\mathsf{sk}}} \leftarrow \mathsf{FHE.Enc}(\mathsf{sk}, \overline{\mathsf{sk}})$*. Then*

1. $\mathsf{lockObf}'$ *is a secure lockable obfuscation scheme, and*
2. *the cycle tester build as specified by Construction 2, from* AFHE *and* $\mathsf{lockObf}'$ *is not ideal cipher annihilation secure.*

Proof (Sketch). The first part is rather straightforward. Since the encryption of $\mathsf{ct}_{\mathsf{FHE},\overline{\mathsf{sk}}}$ is independent of the lock, we change it to an encryption of a uniformly random message given semantic security of the FHE scheme. Then we run the simulator of $\mathsf{lockObf}$.

For the second part of the lemma, consider the following attack. An attacker, obtains the AFHE ciphertexts $\overline{\mathsf{ct}}_k$ for $k \in [\kappa]$ from Definition 12, computes $\mathsf{ct}_k \leftarrow \mathsf{FHE.Eval}(\mathsf{ct}_{\mathsf{FHE},\overline{\mathsf{sk}}}, \mathsf{AFHE.Dec}(., \overline{\mathsf{ct}}_k))$, and evaluates the ideal ciphers on ct_k.

Remark 7 (Extending the Attack). Note that when we additionally include the ciphertext $\mathsf{AFHE.Enc}(\overline{\mathsf{sk}}, \mathsf{sk})$ in the obfuscated circuit above, we can further switch from FHE ciphertexts back to AFHE ciphertexts. In this case, the lockable obfuscator's security requires assuming circular security between AFHE and FHE.

Remark 8 (On the Attack). Note that the adversary in the attack above learns nothing about the secret keys. In particular, the adversary isn't even able to verify whether his computations were correct or not and whether the adversary lives in a world where the obfuscation is simulated or not. Nevertheless, when the lockable obfuscation consists of data that allows performing correct homomorphic operations, the ideal cipher annihilation assumption is broken.

The ideal cipher annihilation security and assumptions like linear-only encryption are based on heuristic beliefs about the underlying cryptosystems' homomorphic capabilities. The example shows that, whenever publishing auxiliary

[4] The idea for the construction is due to an anonymous reviewer.

information that depends on the cryptosystem's secret key, we must argue again that the auxiliary information doesn't help to perform homomorphic operations. In particular, the fact that the auxiliary data does not leak any information on the secret key is not enough of an argument.

Additional Preliminaries The following lemma establishes the notation for trapdoor and preimage sampling that we use later.

Lemma 3 (Trapdoor Sampling). *For dimensions $n \in \mathbb{N}$ and $m \in \mathbb{N}$ and a modulus q, let $(\mathbf{A}, \mathsf{td}) \leftarrow \mathsf{TrapSam}(n, m, q)$, be an algorithm that outputs a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and a trapdoor td, such that $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ is statistically close to the uniform distribution. There is a probabilistic polynomial time algorithm $\mathbf{d} \leftarrow \mathsf{SamPre}(\mathbf{A}, \mathbf{y}, \mathsf{td}, B)$ that on input $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, td, $\mathbf{y} \in \mathbb{Z}_q^n$ and a bound B outputs $\mathbf{d} \in \mathbb{Z}_q^{m \times 1}$ such that $\mathbf{A} \cdot \mathbf{d} = \mathbf{y}$ and $\|\mathbf{d}\|_2 \leq B$.*

The lockable obfuscator computes functions that are in NC^1, such us decryption of LWE tuples. Technically, the obfuscator from [CVW18] computes matrix branching programs that are defined as follows.

Definition 17 (Matrix Branching Program). *Let $\upsilon \in \mathbb{N}$ be the length, $\omega \in \mathbb{N}$ the width of the branching program and $\eta \in \mathbb{N}$ the bit-length of the input. A width-ω, length-υ matrix branching program MBP over η-bit inputs is a tuple*

$$\mathsf{MBP} = (\mathit{inp}, \mathbf{v}, \{\mathbf{M}_{i,b}\}_{i \in [\upsilon], b \in \{0,1\}}, P_0, P_1)$$

where $\mathit{inp} \in [\eta]^\upsilon$, $\mathbf{v} \in \mathbb{Z}_q^\omega$, $\mathbf{M}_{i,b} \in \{0,1\}^{\omega \times \omega}$, and $P_0, P_1 \subset \{0,1\}^{\omega \times \omega}$. Furthermore, the vector $\mathbf{v}$ and the sets P_0 and P_1 satisfy $\mathbf{v} \cdot \mathbf{w} = \mathbf{0}^{1 \times w}$ for all $\mathbf{w} \in P_0$, and $\mathbf{v} \cdot \mathbf{w} \neq \mathbf{0}^{1 \times w}$ for all $\mathbf{w} \in \mathsf{P}_1$.

A matrix branching program MBP computes a function $F : \{0,1\}^\eta \mapsto \{0,1\}$ as

$$F(\mathbf{x}) = \begin{cases} 0 & \text{if } \prod_{i=1}^{\upsilon} \mathbf{M}_{i,\mathbf{x}[\mathit{inp}[i]]} \in P_0 \\ 1 & \text{if } \prod_{i=1}^{\upsilon} \mathbf{M}_{i,\mathbf{x}[\mathit{inp}[i]]} \in P_1. \end{cases}$$

GGH15 Encodings and the Lockable Obfuscation Candidate. As a subroutine, we use the generalized $\mathsf{GGH15}$ encodings.

Construction 7 (Generalized $\mathsf{GGH15}$ Encodings) *Let $\mathsf{TrapSam}$ and SamPre be the algorithms from Lemma 3. Let $\gamma : \mathbb{Z}^{\omega \times \omega} \times \mathbb{Z}^{n \times n} \to \mathbb{Z}^{t \times t}$, be such that $\gamma(\mathbf{M}, \mathbf{S}) \cdot \gamma(\mathbf{M}', \mathbf{S}') = \gamma(\mathbf{M} \cdot \mathbf{M}', \mathbf{S} \cdot \mathbf{S}')$. We define the following encoding function.*

$\mathsf{GGH15Encode}(\gamma, B_{\mathbf{E}}, B_{\mathbf{D}}, [M_{i,b}]_{i \in [\upsilon], b \in \{0,1\}}, [\mathbf{S}_{i,b}]_{i \in [\upsilon], b \in \{0,1\}}, \mathbf{A}_\upsilon)$**:** *This algorithm takes as input a function $\gamma : \mathbb{Z}^{\omega \times \omega} \times \mathbb{Z}^{n \times n} \to \mathbb{Z}^{t \times t}$, bounds $B_{\mathbf{E}}, B_{\mathbf{D}} \in \mathbb{N}$, matrices $\mathbf{M}_{i,b} \in \{0,1\}^{\omega \times \omega}$, matrices $\mathbf{S}_{i,b} \in \mathbb{Z}_q^{n \times n}$ and a matrix $\mathbf{A}_\upsilon \in \mathbb{Z}_q^{t \times m}$, where $n, m, t, \omega \in \mathbb{N}$.*

1. *For $i \in \{0, \ldots, \upsilon - 1\}$ compute $(\mathbf{A}_i, \mathsf{td}_i) \leftarrow \mathsf{TrapSam}(t, m, q)$.*
2. *For $i \in [\upsilon]$ and $b \in \{0,1\}$*

2.1. Compute $\hat{\mathbf{S}}_{i,b} \leftarrow \gamma(\mathbf{M}_{i,b}, \mathbf{S}_{i,b})$.
2.2. Sample $\mathbf{E}_{i,b} \xleftarrow{\$} \mathcal{X}_{\mathbf{E}}^{t\times m}$.
2.3. Compute $\mathbf{D}_{i,b} \leftarrow \mathsf{SamPre}(\hat{\mathbf{S}}_{i,b} \cdot \mathbf{A}_i + \mathbf{E}_{i,b}, \mathsf{td}, B_{\mathbf{D}})$.
3. *Output* $(\mathbf{A}_0, \{\mathbf{D}_{i,b}\}_{i\in[\upsilon],b\in\{0,1\}})$.

Finally, we are ready to describe the obfuscator. The lockable obfuscation construction defines the γ function, as

$$\gamma_{\otimes\mathsf{diag}} : \mathbb{Z}^{w\times w} \times \mathbb{Z}^{n\times n} \mapsto \mathbb{Z}^{(wn+n)\times(wn+n)}, (\mathbf{M}, \mathbf{S}) \mapsto \begin{bmatrix} \mathbf{M}\otimes\mathbf{S} & \mathbf{0} \\ \mathbf{0} & \mathbf{S} \end{bmatrix}$$

We adapt the notation form [CVW18] and for a matrix $\mathbf{A} \in \mathbb{Z}^{t\times *}$, where $t = \eta\cdot\omega\cdot n+n$, we write $\mathbf{A} = \begin{bmatrix}\mathbf{B}\\ \mathbf{C}\end{bmatrix}$, where $\mathbf{B} \in \mathbb{Z}^{\eta\cdot\omega\cdot n\times 1}$ and $\mathbf{C} \in \mathbb{Z}^{n\times 1}$. We write $\mathbf{B}^{[i]}$ to denote the matrix that consists of the rows of $\mathbf{B}$ from $((i-1)\cdot\omega\cdot n+1)$ to $(i\cdot\omega\cdot n)$. Furthermore, as $\mathbf{B}^{[i,j]}$ we denote the matrix that consists of the rows of $\mathbf{B}^{(i)}$ from $((j-1)n+1)$ to (jn).

Construction 8 (Lockable Obfuscator from [CVW18]) *For a circuit* $C : \{0,1\}^\eta \mapsto \{0,1\}^\nu$ *let* $C^{(j)} : \{0,1\}^\eta \mapsto \{0,1\}^\nu$ *be the function that outputs the* j*th output bit of* C*. In other words,* $C^{(j)}(\mathbf{x}) = F(\mathbf{x})[j]$*, for* $j \in [\eta]$*. Let* $\mathsf{MBP}^{(j)} = (\mathsf{inp}^{(j)}, \mathbf{v}^{(j)}, \{\mathbf{M}_{i,b}^{(j)}\}_{i\in[\upsilon],b\in\{0,1\}}, P_0^{(j)}, P_1^{(j)})$ *denote the matrix branching program that computes* $C^{(i)}$*. Assume that there is a tuple* $(\mathsf{inp}, P_0, P_1, \mathbf{v})$ *such that* $\mathsf{inp} = \mathsf{inp}^{(j)}$*,* $P_0 = P_0^{(j)}$ *and* $P_1 = P_1^{(j)}$*, and* $\mathbf{v} = \mathbf{v}^j$ *for all* $j \in [\nu]$*.*

$\mathsf{Obf}(\lambda, C, \mathsf{lock})$**:** *This* **PPT** *algorithm takes as input a security parameter* $\lambda \in \mathbb{N}$*, a circuit* $C \in \mathcal{C}_{\eta,\nu,\xi}$*, where* $\eta, \nu, \xi = \mathsf{poly}(\lambda)$*, and a lock string* $\mathsf{lock} \in \{0,1\}^\nu$*.*

1. *Set the bounds* $B_{\mathbf{S}}$*,* $B_{\mathbf{D}}$ *and* $B_{\mathbf{E}}$ *such that the ratio* q/B *is "small", where* $B \le \upsilon\cdot B_{\mathbf{E}}\cdot t\cdot(\sqrt{t}\cdot\sqrt{m}\cdot B_{\mathbf{S}}\cdot B_{\mathbf{D}})^{\upsilon-1}$*.*
2. *For* $i \in [\upsilon]$ *and* $b \in \{0,1\}$ *sample* $\mathbf{S}_{i,b} \xleftarrow{\$} \mathcal{X}_{B_{\mathbf{S}}}^{n\times n}$*.*
3. *Set* $\mathbf{A}_\upsilon = \begin{bmatrix}\mathbf{B}\\ \mathbf{C}\end{bmatrix} \in \mathbb{Z}_q^{t\times 1}$*, where* $\mathbf{B} \xleftarrow{\$} \mathbb{Z}_q^{\eta\cdot\omega\cdot n\times 1}$ *and* $\mathbf{C} \leftarrow \sum_{i\in[\nu]} \mathbf{B}^{[i,(2-\mathbf{y}[i])]} \in \mathbb{Z}_q^{n\times 1}$*.*
4. *For* $i \in [\upsilon]$ *and* $b \in \{0,1\}$*, define* $\mathbf{M}_{i,b} \in \{0,1\}^{\eta\cdot\omega\times\eta\cdot\omega}$ *as a diagonal block matrix with* $\mathbf{M}_{i,b}^{(j)} \in \{0,1\}^{\omega\times\omega}$ *in its* j*th diagonal. In other words,* $\mathbf{M}_{i,b} = \mathsf{diag}(\mathbf{M}_{i,b}^{(1)}, \ldots, \mathbf{M}_{i,b}^{(\eta)})$*.*
5. *Set* $\mathbf{M} = [\mathbf{M}_{i,b}]_{i\in[\upsilon],b\in\{0,1\}}$ *and* $\mathbf{S} = [\mathbf{S}_{i,b}]_{i\in[\upsilon],b\in\{0,1\}}$*.*
6. *Compute*

$$(\mathbf{A}_0, [\mathbf{d}_{i,b}]_{i\in[\upsilon],b\in\{0,1\}}) \leftarrow \mathsf{GGH15Encode}(\gamma_{\otimes diag}, B_{\mathbf{E}}, B_{\mathbf{D}}, \mathbf{M}, \mathbf{S}, \mathbf{A}_\upsilon)$$

7. *Set* $\mathbf{J} \leftarrow [\mathbf{v}, \mathbf{v}, \ldots, \mathbf{v}, -1] \otimes \mathbf{I}^{n\times n} \in \{0,1\}^{n\times t}$*.*
8. *Return* $\tilde{C} = (\mathbf{J}\cdot\mathbf{A}_0, [\mathbf{d}_{i,b}]_{i\in[\upsilon],b\in\{0,1\}})$*.*

$\mathsf{Eval}(\tilde{C}, \mathbf{x})$: *Takes as input an obfuscated circuit* $\tilde{C} = (\mathbf{J}\cdot\mathbf{A}_0, [\mathbf{d}_{i,b}]_{i\in[\upsilon],b\in\{0,1\}})$ *and input* $\mathbf{x} \in \{0,1\}^\eta$*.*

1. *Compute* $\mathbf{d} = \prod_{i\in[\eta]} \mathbf{d}_{i,\mathbf{x}[i]}$*.*
2. *If* $||\mathbf{J}\cdot\mathbf{A}_0\cdot\mathbf{d}||_2 \le B$*, return* 1*. Otherwise return* $\perp$*.*

On Security of the Candidate Instantiation. First, we emphasize that we consider the candidate instantiation as given by Construction 2 with the lockable obfuscator limited only to the GGH15 encoding. In particular, we do need to switch AFHE ciphertexts to any other fully homomorphic encryption scheme as AFHEs decryption function is in NC^1 and can directly be implemented by the matrix branching program. On the other hand, we may add a key switching key to the last level of the AFHE scheme and instantiate the obfuscation for the key switching cryptosystem's decryption circuit. However, we note that we must be careful to instantiate the key switching key properly, i.e., do not publish any further relinearization or key switching keys and choose the error magnitudes to allow only computation necessary for correctness. In particular, it is enough that the ciphertexts constituting the key switching key are only additively homomorphic.

Note that semantic security of the cycle tester holds, given DVBB of the lockable obfuscation and semantic security of the base AFHE. Furthermore, note that the keys in the ideal cipher annihilating security definition are chosen independently of the lock. Following the same reasoning as for semantic security, we can show that a world where the obfuscated program $\tilde{C}$ is simulated is indistinguishable from the real world. In other words, $\tilde{C}$ gives no information on the AFHE secret key. Thus we believe that an adversary cannot perform inconsistent operations on the AFHE ciphertext itself but must switch from AFHE ciphertexts to ciphertexts of an alternative cryptosystem based on the information that constitutes $\tilde{C}$.

We argue that it seems hard to use the GGH15 encodings to perform any meaningful homomorphic computation. On a high level, we choose the parameter and error magnitudes of the GGH15 encodings only to support the matrix branching program's computation. Furthermore, the encodings are not designed to be ever be decrypted. Instead, when evaluated on $\mathbf{x}$ such that $C(\mathbf{x}) = \mathsf{lock}$, we have that $||\mathbf{J} \cdot \mathbf{A}_0 \cdot \prod_{i\in[\eta]} \mathbf{d}_{i,\mathbf{x}[i]}|| = \prod_{i\in[\eta]} \mathbf{S}_{i,\mathbf{x}[i]} \cdot \mathbf{0}^n + \mathbf{E}^* = \mathbf{E}^* \leq B$ from correctness of the obfuscator. When evaluated on $\mathbf{x}$ such that $C(\mathbf{x}) \neq \mathsf{lock}$, we have $\mathbf{J} \cdot \mathbf{A}_0 \cdot \prod_{i\in[\eta]} \mathbf{d}_{i,\mathbf{x}[i]} = \prod_{i\in[\eta]} \mathbf{S}_{i,\mathbf{x}[i]} \cdot \mathbf{A}_h^{\mathcal{I}} + \mathbf{E}^* = \mathbf{E}^*$, where $\mathbf{A}_h^{\mathcal{I}}$ is the sum of independent uniformly random vectors over $\mathbb{Z}_q^n$.

Looking closer, we argue that the encodings do not have the necessary structure to support meaningful homomorphic computation. Remind that $\mathbf{d}_{i,b}$ are preimages of $\hat{\mathbf{S}}_{i,b} \cdot \mathbf{A}_i + \mathbf{E}_{i,b}$. Furthermore, $\mathbf{A}_i$ are trapdoor matrices, $\mathbf{E}_{i,b}$ are small norm matrices, and $\hat{\mathbf{S}}_{i,b} = \gamma(\mathbf{M}_{i,b}, \mathbf{S}_{i,b})$ with $\mathbf{S}_{i,b}$ being also small norm matrices without any other special structure. According to the security analysis, [CVW18], we treat the $\mathbf{A}_i$ matrices as the secret keys of LWE tuples. Observe that it is already unclear how to start any meaningful homomorphic computation from $\mathbf{d}_{i,b}$ since, as mentioned, those are trapdoor preimages. The tuples $\hat{\mathbf{S}}_{i,b} \cdot \mathbf{A}_i + \mathbf{E}_{i,b}$ may be viewed as matrices of LWE tuples $b = \langle \hat{\mathbf{S}}_{i,b}[j,*], \mathbf{A}_i[*,k] \rangle + \mathbf{E}_{i,b}[j,k]$, for $j \in [t]$ and $k \in [m]$, but without explicit access to $\hat{\mathbf{S}}_{i,b}[j,*]$. Note that from these tuples, we don't have any key switching keys. Furthermore, the tuples do not possess any special structure, like for example, the GSW [GSW13]

cryptosystem. Finally, the error magnitude of $\mathbf{E}_{i,b}$ isn't chosen to support any further homomorphic computation.

As an end note, we observe that, for our application to witness encryption, $\tilde{C}$ only makes a difference to the adversary A if A manages to test whether there is a key cycle in the AFHE ciphertext after homomorphically evaluating the garbled circuit or not. Importantly, we would need to strengthen the attack from the beginning of this section to the case described in Remark 7. In our candidate instantiation with $\mathsf{GGH15}$ encodings only, the ability to re-encrypt from $\mathsf{GGH15}$ encodings back to AFHE would hint at a possibility to exploit $\mathsf{GGH15}$ as a bootstrapping algorithm. Specifically, we could obtain fully homomorphic encryption instead of level homomorphic encryption without circular security. Note that from the proof of Theorem 7, we have that semantic security for this construction doesn't require circular security.